MASTER EN SCIENCES INFORMATIQUES

Conception et implémentation d'un afficheur de partitions musicales en C++ et en Java

Lejoly, Luc

Award date:
1997

CONCEPTION ET IMPLÉMENTATION D'UN AFFICHEUR DE PARTITIONS MUSICALES EN C++ ET EN JAVA

Mémoire réalisé par Luc **Lejoly**

en vue de l'obtention du diplôme de « Maître en Informatique »,

suite à son stage en entreprise à Logi+ SA, 6 rue de Stockholm, Strasbourg, France,
du 1er octobre 1996 au 31 janvier 1997.

Promoteur : Monsieur Claude **Cherton**

Maître de stage : Monsieur Éric **Bishoff**

Abstract :
This thesis results from a four-month-training period at Logi+ (Strasbourg, France). It consists of explanations about the conception and implementation of the « musical score display » sub-project in a musical database manager. This work is not simply meant as a thesis, its purpose is also to help understand and maintain the program.

Keywords : musical score, object-oriented, C++, Java.

Remerciements

Grand merci à Monsieur et Madame Sturm et à toute l'équipe de Logi+ pour leur accueil si chaleureux, spécialement à Éric Bishoff qui m'a donné tant d'attention et m'a fait partager sa grande expérience.

Également à Monsieur Cherton, pour ses conseils avisés sans lesquels ce mémoire n'aurait pas décollé et aurait été envahi par des ambiguïtés et autres contradictions.

Merci à mes parents, à Jean-Pol Albert, à Marie-Thérèse et Joseph Bayet et à Vinciane Devuyst pour leur collaboration très précieuse.

Enfin, une pensée à toutes les personnes qui, d'une manière ou d'une autre, m'ont supporté, dans tous les sens du terme, pendant ces cinq années.

Plan

Table des illustrations

Introduction

Créée en 1983, Logi+[1] est une petite entreprise qui développe et distribue des logiciels de gestion documentaire et de gestion de bibliothèque. Actuellement, elle emploie six personnes dont deux s'occupent du développement de logiciels. Le reste du staff est employé aux activités de support aux programmeurs (documentation, graphisme), aux activités économiques (comptabilité, marketing) et est avant tout au service des clients. Cette relation de service est très importante dans la politique commerciale de Logi+. Sur le marché des PMÆ qu'elle affectionne particulièrement, cela lui donne en effet un net avantage sur des concurrents de plus grande taille qui ne peuvent offrir un service aussi personnalisé.

Logi+ développe principalement quatre logiciels : GesBib, logiciel de gestion documentaire ; BiBal, logiciel de gestion documentaire et de gestion de prêt; InfoMusic, logiciel de gestion documentaire, permettant de gérer des informations musicales comme information documentaire (recherche sur base d'informations musicales, affichage d'informations musicales,...) et BibliMuse, logiciel regroupant les possibilités de BiBal et d'InfoMusic. La taille de ces programmes exécutables (en environnement DOS) est d'environ 1 Mo, celle des sources (quelque 150 000 lignes de code) doit atteindre 2,5 Mo. Les deux logiciels InfoMusic et BibliMuse servent essentiellement de vitrine à la firme, ils ne représentent qu'un faible pourcentage du chiffre d'affaires. Parmi les utilisateurs les plus prestigieux de ces logiciels, citons le Centre Polyphonique d'Alsace (Munster), le Centre Polyphonique de Bourgogne (Auxerre), la Bibliothèque Nationale (Paris), la Maison Claude Debussy (Saint Germain en Laye), l'International Center for Choral Music (Namur), le Centrum für Deutsche und Internationale Chormusik (Selters - Allemagne), le Centre de Documentation musicale A Coeur Joie Belgique (Ciney) et les Éditions musicales Schola Cantorum (Fleurier - Suisse). Les quatre produits sont destinés à tout gestionnaire de bibliothèque ou de fond documentaire, de la bibliothèque scolaire (quelques centaines de fiches) à des bases de données de plus de 100 000 fiches. Les clients gèrent, en moyenne, des fichiers de 5000 fiches. Musica qui compte, à ce jour, quelque 67 000 fiches est la plus importante base de données gérée avec InfoMusic.

[1] 6, rue de Stockholm à Strasbourg

Excepté Musica, qui est une organisation au rayonnement mondial, tous les clients de Logi+ sont européens, principalement Français, Allemands et Suisses.

Ces logiciels sont aisément adaptables par l'utilisateur. Ils lui permettent de définir des stratégies de saisies, des formulaires de recherche, des formats de présentation de données adaptés à ses besoins.

Ils sont multilingues. Actuellement, des versions en langues française, anglaise, allemande et espagnole existent ; une traduction néerlandaise est en cours. En plus du choix de la langue utilisée dans l'interface, ces logiciels permettent la création de thesaurus qui associent différentes traductions de chaque mot-clé. Cela permet, lors d'une recherche, de trouver des fiches où les mots recherchés sont exprimés dans une autre langue.

Les responsables de Logi+ ont toujours anticipé le développement des outils télématiques. Très tôt, une solution de consultation, de recherche à distance dans des bases de données par Minitel a été réalisée. Plus récemment, ces possibilités sont offertes via un serveur Web.

Musica International est le plus important des utilisateurs d'InfoMusic et de BibliMuse. L'histoire de ces logiciels et leur succès sont fortement liés au développement de cette organisation. Il me semble important de présenter brièvement cette organisation et son évolution.

Musica est née en 1983 à l'initiative du Centre d'Art Polyphonique d'Alsace (CAPA). Ce projet de base de données de partitions chorales se développe à l'échelle régionale jusqu'en 1990, année où le Centre International pour la Musique Chorale (CIMC) de Namur va s'y associer pour, ensemble, « créer une banque de données exhaustive du répertoire de musique chorale du monde entier »[2]. L'entreprise serait utopique si elle n'avait comme moteur des personnes très motivées. Dès le début, une équipe dynamique comprenant, entre autres, messieurs Jean Sturm et Jean-Claude Wilkens, s'est formée et a porté ce projet à bout de bras.

Depuis, Musica International s'agrandit régulièrement. Il compte à ce jour des membres venant de 11 pays[3]. Ce sont principalement des centres de documentation musicale et des maisons d'édition. Les membres s'engagent à encoder les données de leur fond musical et à les communiquer en vue de leur intégration à la base de données Musica. En contrepartie, ils sont autorisés à utiliser les logiciels InfoMusic et BibliMuse et ont accès aux données de la base Musica. Les membres se réunissent régulièrement lors de sessions pendant lesquelles ils mettent à jour la base de données Musica et la redistribuent.

Aujourd'hui, Musica International se trouve à un tournant de son histoire. Son importance et ses coûts de fonctionnement sont devenus tels qu'elle doit devenir une entreprise à part entière : louer l'accès à sa base de données et rémunérer ses responsables et ses

[2] leurs propres termes

[3] France, Belgique, Allemagne, Grande-Bretagne, Pays-Bas, Norvège, Hongrie, Estonie mais aussi Etats-Unis, Canada, Venezuela et Argentine

employés. En ignorant les problèmes organisationnels possibles, pour mener à bien ces changements, il faudra relever deux défis majeurs : arriver à une grande cohérence et pertinence des données de la base et fournir des logiciels ergonomiques et efficaces.

Inclure des informations concernant la musique dans une base de données n'est pas très utile s'il est impossible de les « montrer » d'une manière ou d'une autre à l'utilisateur. Il existe chez l'homme deux canaux propices à la réception de données musicales : l'ouïe et la vision. Dans le développement du logiciel InfoMusic, l'exploitation de ces deux canaux fait l'objet de deux projets distincts. Le projet qui nous occupe ici est celui qui permet de visualiser les données musicales dans InfoMusic.

Ce mémoire a été réalisé sur base du travail réalisé pendant un stage de quatre mois au siège de Logi+. Ce travail a consisté en la réalisation de la partie « affichage d'un champ musical » de la nouvelle version d'InfoMusic. Dans un premier temps, une version C++ a été développée, ensuite, celle-ci a été traduite en une applet Java. Ce document s'articule comme suit : dans le premier chapitre, nous introduisons les concepts musicaux dont la connaissance est nécessaire à la bonne compréhension des chapitres suivants ; le second regroupe les diverses parties du « cahier des charges » ; le troisième chapitre présente les classes d'objets conçues lors de la réalisation ; l'explication de leur implémentation fait l'objet du quatrième chapitre ; les deux derniers chapitres sont consacrés respectivement à la version Java de l'afficheur et aux améliorations qu'il serait possible d'effectuer à l'avenir sur ces deux projets (C++ et Java).

Chapitre 1. Concepts musicaux

Dans ce chapitre, nous introduisons les concepts musicaux utiles à la compréhension des chapitres suivants. Notre but n'est pas ici de donner un aperçu exhaustif de la théorie des notations musicales, mais d'introduire le lecteur au monde du musicien à qui le programme est destiné.

Ce travail ne traite que de la musique dite « classique », à distinguer d'autres musiques telles que les musiques jazz, orientale ou électroacoustique dont les possibilités et les règles sont différentes et rarement formalisées. Par la suite, le terme « musique » se restreindra à ce concept.

Dans une pièce musicale, on distingue deux aspects : la mélodie et l'harmonie. La mélodie est l'aspect horizontal de la musique, la succession dans le temps de différents sons. L'harmonie est son aspect vertical, simultané, représenté par des accords. Nous nous intéresserons ici plus particulièrement aux mélodies car l'harmonie n'est pas encore gérée par InfoMusic[4].

1. Représentation du son

Les musiciens représentent les sons par des symboles qu'ils appellent **notes**. La note permet par sa position et son dessin de caractériser à la fois la hauteur et la durée d'un son.

1.1 La hauteur du son

Du point de vue physique, le caractère plus ou moins aigu d'un son dépend de sa fréquence[5] : plus la fréquence d'un son est grande, plus il est aigu. L'intervalle entre deux notes de fréquence double est appelé **octave**. Pour des raisons acoustiques, cet intervalle

---

[4] des informations complémentaires concernant la théorie musicale peuvent être trouvées dans [HIND,86], concernant l'histoire des notations musicales dans [MACH,52] ; enfin, [VIGN,87], dictionnaire de la musique très précis, peut servir de glossaire.

[5] en physique : nombre de vibrations par unité de temps dans un phénomène périodique.

sert de base à l'échelle des notes. Actuellement, l'octave est divisée en douze parties égales appelées **demi-tons**. Le clavier du piano est composé de touches noires et de touches blanches. Une séquence de couleurs se répète toutes les douze touches. Une octave sépare la première note d'une série de la première de la série suivante. L'intervalle entre deux sons correspondant à deux touches adjacentes est un demi-ton. Les notes ne se répartissent pas uniformément dans l'octave. Elles sont au nombre de sept (de la plus grave à la plus aiguë : do, ré, mi, fa, sol, la et si) et se répètent d'octave en octave. Ce sont les touches blanches du piano. L'intervalle entre deux notes est d'un ton sauf entre mi-fa et si-do où il n'est que d'un demi-ton.

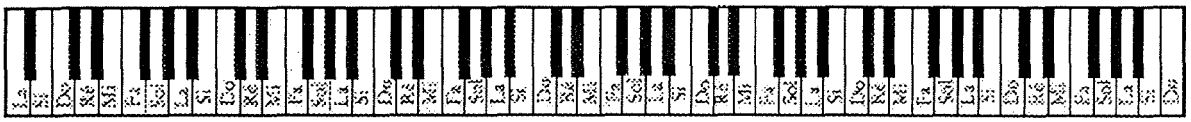


Figure 1: clavier de piano[6]

Les notes se dessinent sur une **portée**. Une portée se compose de cinq lignes horizontales. Sur chaque ligne ou dans chaque interligne, il est possible de dessiner une note. Des barres parallèles à la portée sont ajoutées localement afin de faciliter la mesure de la hauteur d'une note trop haute ou trop basse pour être dessinée dans la portée.

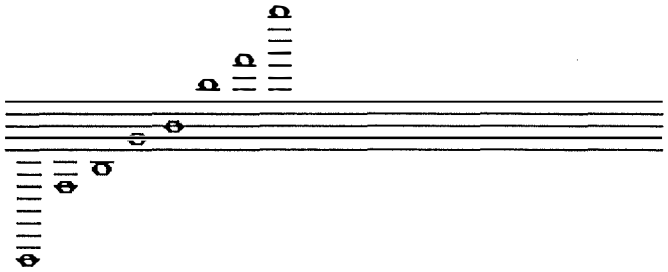


Figure 2: portée et notes

Il existe un grand nombre d'instruments, aux tessitures[7] si diverses qu'il faudrait environ 26 lignes de portée pour dessiner toutes les notes possibles. On fait donc varier la position de la portée verticalement sur l'échelle des notes. Le rôle de la **clef** est de fixer la hauteur de la portée en associant une note à une ligne. Il en existe de trois sortes (clefs de fa, d'ut et de sol) liant respectivement les notes fa, do et sol à une ligne de la portée. L'usage a restreint l'utilisation de la clef de fa à la 3ème ou 4ème ligne et la clef de sol à la 1ère ou 2ème ligne. La clef d'ut peut être employée sur les cinq lignes. Sur la figure suivante, la note *do* est représentée dans les différentes clefs. En pratique, les clefs de fa sont associées aux instruments graves (basse, trombone, contrebasse...), les clefs d'ut à

[6] extrait de [PIER,87]

[7] tessiture : intervalle dans lequel une voix chante ou un instrument sonne, dans lequel un morceau, une partie vocale ou instrumentale sont écrits

des instruments intermédiaires (violoncelle, alto, basson...) et les clefs de sol à des instruments aigus (violon, trompette, clarinette, flûte...). La clef utilisée est rappelée au début de chaque portée. Un changement de clef peut survenir n'importe où dans la mélodie : la nouvelle clef est simplement dessinée devant la première note à lire différemment.

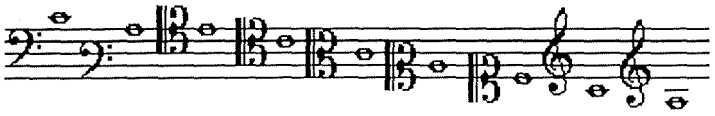


Figure 3: les différentes clefs

La hauteur d'une note peut être modifiée par des **altérations**. Les altérations usuelles sont le **bémol** qui baisse la note d'un demi-ton et le **dièse** qui la hausse d'un demi-ton. Les règles de composition exigent parfois qu'une note soit haussée de deux demi-tons. Même si le son de la note résultante s'entend comme une note non altérée, le musicien utilise des altérations doubles (double bémol et double dièse) dont l'effet est de baisser ou de hausser la note de deux demi-tons. Un sol double dièse a la même fréquence qu'un la mais est représenté différemment. Le **bécarre** n'est pas un signe d'altération au sens propre mais plutôt un signe d'annulation qui rend à la note sa hauteur normale[8] ; en d'autres termes, il interrompt l'effet d'une altération antérieure.

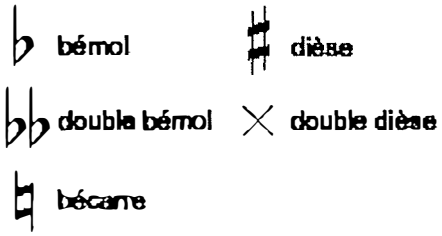


Figure 4: les altérations

On distingue deux usages d'altérations : les altérations dites **accidentelles** et les altérations dites d'**armature**.

L'altération accidentelle se dessine à gauche d'une note. Elle affecte toute note de même hauteur, à venir dans la mesure[9] où elle se trouve.

Les altérations d'armature sont dessinées en début de portée (à droite de la clef) sur une ligne ou dans un interligne. Elles définissent le ton (la tonalité) de la mélodie et sont aussi appelées altérations **à la clef** ou **armature**. Elles s'appliquent à toutes les notes de la portée situées à cette hauteur et aux notes de même nom que ces dernières mais

[8] les musiciens parlent d'état naturel de la note
[9] concept de mesure : cfr infra

d'octaves différentes. L'armature est composée de 0 à 7 bémols ou de 0 à 7 dièses. La suite des positions des altérations dans une armature doit être un préfixe[10] de [fa, do, sol, ré, la, mi, si] pour les dièses et de [si, mi, la, ré, sol, do, fa] pour les bémols.

La tonalité peut changer en cours de mélodie. Divers cas de figure peuvent alors se présenter selon la composition de l'ancienne et de la nouvelle armature. Ils sont tous illustrés sur la figure suivante. On peut remarquer les deux situations d'exception où l'ajout de bécarres est nécessaire pour annuler des altérations de l'ancien ton.

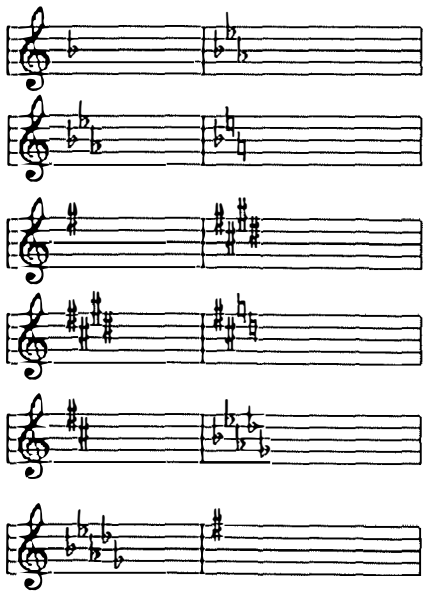


Figure 5: changements de tons

1.2 La durée du son

La durée[11] de la note est principalement déterminée par son dessin. La figure suivante représente des dessins de notes. Ces notes sont classées par ordre décroissant de durée ;

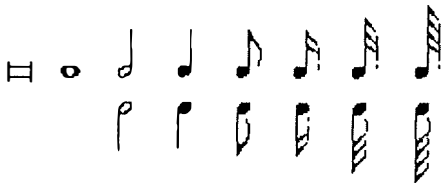


Figure 6: durées de notes : carrée, ronde, blanche, noire, croche, double croche, triple croche et quadruple croche

chaque note est deux fois plus longue que sa voisine de droite. Cette liste de dessins peut être complétée à droite si nécessaire. Chaque dessin peut se décomposer en trois

[10] au sens mathématique du terme : préfixe = ensemble d'éléments débutant une suite.
[11] les musiciens parlent de rythme de la note

parties : le corps de la note, la hampe et les moustaches. Seul le corps de la note se retrouve obligatoirement. En effet, c'est la position du corps de la note par rapport à la portée qui détermine sa hauteur. L'orientation de la hampe dépend de la position de la note par rapport à la portée. Elle sera orientée vers le haut si la note est basse, vers le bas si la note est haute.

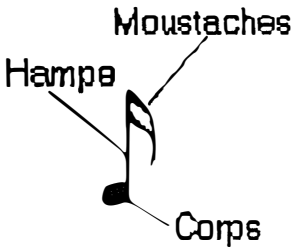


Figure 7: parties d'une note

Des **points** peuvent être ajoutés à droite du corps de la note. Un point augmente la durée de la note à laquelle il se rapporte de d * $(½)^p$ où d est la durée de la note et p est la position du point. Le premier point ajoute ½ fois la durée, le $2^{ème}$, ¼ de fois, ...

Deux notes de même hauteur peuvent être reliées par une courbe (liaison). Elles n'en forment alors plus qu'une dont la durée est la somme des durées des deux notes. Par la suite, nous appellerons ces liaisons, **liaisons de durée**.

Une ligne verticale placée régulièrement en travers de la portée est appelée **barre de mesure**. Il s'agit d'un repère visuel, destiné à éviter de trop grandes difficultés d'exécution[12]. Ce qui se trouve entre deux barres de mesure consécutives constitue une **mesure**. Si la durée d'une mesure est fixe, il est d'usage de le mentionner par une **indication de mesure** sous forme de fraction au début de la première mesure. Le dénominateur de la mesure représente une division de la ronde, et le numérateur le nombre de ces divisions contenues par mesure. Par exemple, une mesure 3/4 est formée par trois quarts de ronde, soit trois noires. Des abréviations sont parfois utilisées en remplacement d'indications de mesures courantes : C pour 4/4, C barré pour 2/2, 2 pour 2/1, ... Avant le $XVII^e$ siècle, les durées de mesure étaient souvent variables ; on n'indiquait donc pas de mesure. Il faut encore noter que dans certaines pièces anciennes, on rencontre parfois des mesures dont le contenu ne correspond pas à l'indication de mesure.

[12] pour les musiciens, exécuter une pièce signifie la chanter ou la jouer avec un instrument

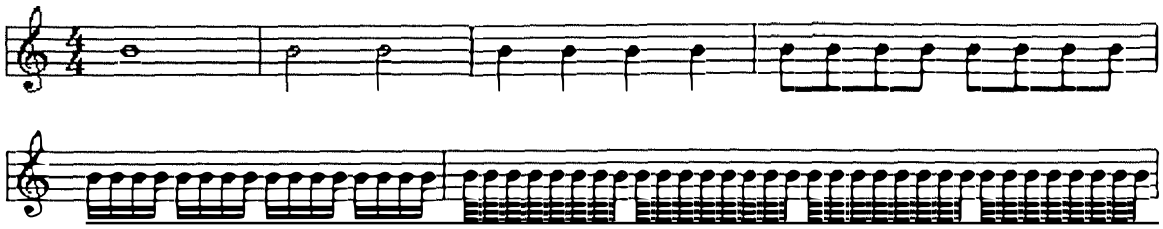


Figure 8: mesure 4/4 et différents contenus

Les mesures se composent de temps : une mesure 4/4 contient 4 temps ; une mesure 2/2, 2 temps ; une mesure 9/8, 3 temps, ... Une mesure est dite **binaire** si la division des temps se fait par deux. Dans une mesure 4/4, l'unité de temps est la noire qui se divise en deux croches, 4/4 est donc une mesure binaire. Une mesure est dite **ternaire** si la division des temps se fait par trois. Dans une mesure 9/8, l'unité de temps est la noire pointée qui se divise en trois croches, 9/8 est donc une mesure ternaire.

Il est d'usage de grouper les notes d'une mesure par temps. Les moustaches des notes sont alors remplacées par des lignes parallèles reliant les notes d'un même temps (cfr figure ci-après).

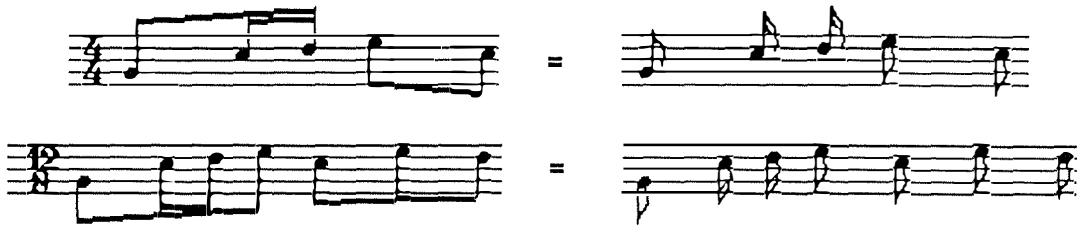


Figure 9: notes liées par temps

Des musiciens ont très vite ressenti le besoin d'émuler des divisions de temps ternaires dans des mesures binaires et vice versa. Les **n-olets** leur permettent cette fantaisie. Il s'agit d'un groupe de notes reliées par un crochet et assorties d'un nombre (le n du n-olet). Les deux n-olets les plus utilisés sont le triolet (n=3), utilisé dans les mesures binaires, dont la durée des notes est multipliée par 2/3 (trois notes sont jouées sur le temps de deux) et son inverse, le duolet (n=2), utilisé dans les mesures ternaires, dont la durée des notes est multipliée par 3/2. Il existe également des quartolets, quintolets, sextolets, septolets..., dont la définition est moins précise. Par exemple, un quintolet peut s'interpréter comme cinq notes se jouant sur le temps de quatre, ou cinq notes se jouant sur le temps de six.

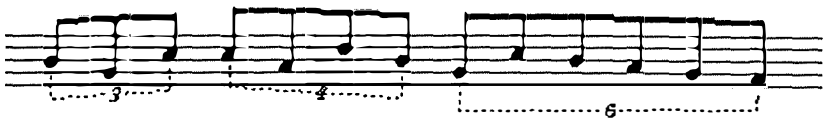


Figure 10: triolet, quartolet et sextolet

Le **tempo** indique la vitesse d'exécution de la pièce. Il s'indique au-dessus de la première portée par une équation « *dessin de note = nombre* ». Celle-ci signifie qu'en une minute l'exécutant chante ou joue *nombre* notes dont le dessin est le membre de gauche. La figure ci-dessous signifie que 144 noires doivent être jouées en une minute.

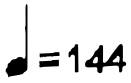


Figure 11: indication de tempo

1.3 Absence de son

L'absence de son dans une mélodie est représentée par des signes particuliers appelés **silences**. La figure suivante montre les silences dont les durées correspondent aux durées des notes de la figure 6. La liste des silences peut également être étendue vers la droite selon les besoins. Comme les notes, les silences peuvent être pointés, liés et faire partie de n-olets.

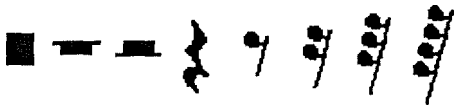


Figure 12: les silences : double pause, pause, demi-pause, soupir, demi-soupir, quart de soupir, huitième de soupir et 16ème de soupir

1.4 Emission du son

Les notes d'une mélodie peuvent être interprétées de différentes manières dont voici quelques exemples : une note accentuée commence puissamment et s'affaiblit progressivement, une note soutenue garde une puissance constante, des notes liées sont exécutées sans interruption entre chaque note, des notes lourées sont liées et les premières notes de chaque temps sont un peu accentuées.

Il existe différentes notations indiquant ces manières de jouer. Nous ne nous intéresserons ici qu'à la **liaison** (notes liées) qui est la seule à être gérée par InfoMusic. Ces liaisons sont dites mélodiques par opposition aux liaisons de durée (cfr plus haut). Elles relient deux notes de hauteur quelconque et séparées par un nombre quelconque de notes (appelées notes liées). Une liaison peut s'étendre sur plusieurs portées. Dans ce cas, elle

est décomposée en sous-liaisons (une par portée). On reconnaît une sous-liaison au fait qu'elle débute en deçà de la première note de la portée et/ou se termine au-delà de la dernière note de la portée.

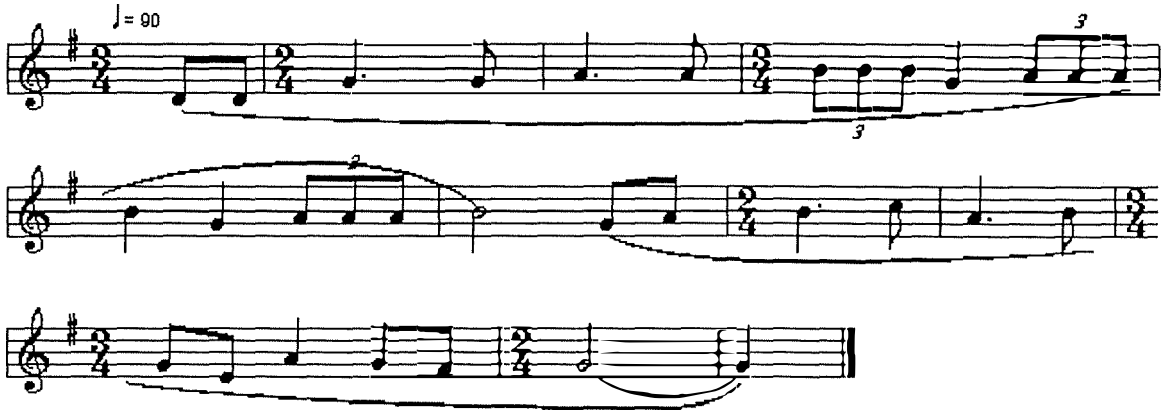


Figure 13: exemple de mélodie complexe avec sous-liaisons

Chapitre 2. Analyse des besoins

1. Existant

1.1 Afficheur

La version précédente d'InfoMusic permettait déjà la visualisation de la musique. Vous pouvez observer l'affichage d'un thème musical sur la figure suivante. L'étude de cet afficheur met en lumière quelques défauts importants :

1) La taille de la mélodie visible est limitée à la largeur de l'écran. Si la mélodie dépasse du côté droit de l'écran, l'afficheur ne crée pas une nouvelle portée, la partie supplémentaire est simplement invisible ;
2) La faible résolution de dessin ne permet pas beaucoup de précision dans les graphiques des notes, de la clef, ... ;
3) Les notes ne sont pas reliées par temps, ce qui rend la lecture difficile et ralentit la détection d'erreurs d'encodage. En effet, si les notes sont reliées par temps, la présence d'une note superflue ou l'absence d'une note nécessaire à la complétude d'un temps se voit très facilement, l'isolement d'une note attirant l'attention du musicien avisé.

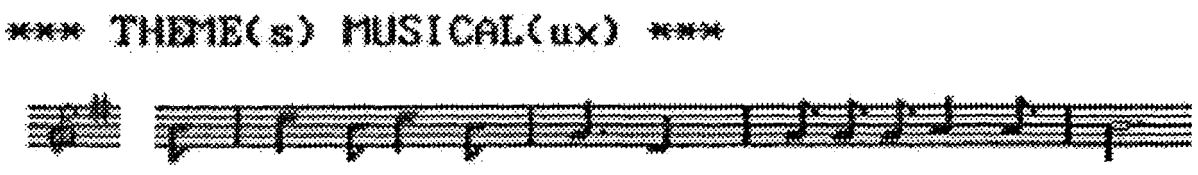


Figure 14: ancienne version de l'afficheur

1.2 Ancien codage

Le codage de la musique dans InfoMusic possède toute une histoire. Au tout début, seule la hauteur des notes de la mélodie était encodée ; ensuite le codage a permis

l'indication des durées, des altérations, des mesures, etc. Les paragraphes suivants énumèrent les possibilités de codage qu'offre la version précédente d'InfoMusic.

La tessiture acceptée s'étend du do2[13] au si6. Les durées des notes sont d'une précision de l'ordre de la triple croche. Il est possible d'altérer une note avec une des trois altérations courantes (bémol, dièse et bécarre).

Les notes se lisent toujours en clef de sol 2ème ligne. La mesure, l'armature et le tempo sont définis en début de mélodie et ne peuvent changer en cours de mélodie.

Des triolets et duolets peuvent être encodés. Ils sont toujours formés respectivement de trois notes de même durée et de deux notes de même durée.

Des liaisons permettent de relier deux notes consécutives. Ces liaisons sont supposées être des liaisons de durée (reliant des notes de même hauteur). Elles apparaissent donc horizontales, à hauteur de la première note. Aucune vérification n'étant réalisée sur la hauteur des notes liées, les utilisateurs ont utilisé ce moyen pour exprimer des liaisons mélodiques qui ne sont aucunement permises autrement.

L'ancien codage, à code de longueur variable, était le résultat des améliorations successives intervenues lors des différentes version d'InfoMusic. Il était devenu ardu à lire. La décision de le refondre en un nouveau codage, aux plus larges possibilités musicales, a alors été prise.

2. Contraintes fonctionnelles

Des études préalables ont été réalisées par Logi+. Elles portaient, d'une part sur les besoins des utilisateurs actuels et potentiels d'Infomusic ; d'autre part, sur le rapprochement du codage de la musique avec la norme MIDI[14] afin de faciliter les traductions et la portabilité des bases de données musicales. Le format MIDI, initialement défini pour interpréter la musique sur des périphériques musicaux (synthétiseurs), n'est pas adapté à l'édition de partitions. Il n'a donc pas été pris tel quel comme codage de mélodie dans la base de données.

Un nouveau codage a été le fruit de ces études menées par Messieurs Bishoff et Sturm de Logi+. Il constitue la spécification des données que l'afficheur musical aura à traiter. La première partie de cette section sera consacrée à la présentation de ce code. Les résultats du traitement de l'afficheur sont déterminés en grande partie par les usages énon-

[13] cette notation identifie une note : il s'agit de son nom suivi du numéro de son octave (du plus grave au plus aigu)

[14] Musical Instrument Digital Interface : protocole largement accepté et utilisé par les musiciens et les compositeurs depuis sa conception vers 1982. C'est une méthode très efficace pour représenter des informations d'interprétation de la musique (cfr [LIPS,89], [HECK,95] ou [MCQU,95]). Un format de fichier en est dérivé (le Standard MIDI File) : [SMF,96].

cés au chapitre précédent, que nous ne rappellerons pas ici, et par des paramètres définis par l'utilisateur que nous présenterons dans la seconde partie de cette section.

2.1 Nouveau codage

Une mélodie est représentée sous forme d'une suite de paquets[15] de longueur fixe (3 octets). Le premier paquet doit être un paquet d'en-tête et le dernier un paquet de fin.

Chaque paquet comprend deux parties : un préfixe de taille variable identifiant son type et une partie de données se décomposant généralement en parties plus petites. Il existe dix types de paquets dont voici les préfixes et les noms :

Préfixes	**Noms**
0	Note
1000	Silence
10010	Tonalité
10011	Clef
1010	Mesure
1011	Barre de mesure
1100	En-tête
1101	n-olet
1110	Liaison
1111	Fin

Dans les sous-sections suivantes, nous passons en revue ces différents paquets. La première ligne de chaque sous-section donne un résumé du format de ces paquets. Chaque caractère de ce résumé représente un bit.

2.1.1 Paquet « Note »

paquet : `0hhh hhhh cc00 dddd dddd dddd`

Ce paquet peut être décomposé en trois parties. Nous voyons d'abord 7 bits codant la hauteur de la note. Les hauteurs possibles vont du do-2 (0x00) au sol8 (0x7f) par pas d'un demi-ton.

La deuxième partie (2 bits) caractérise l'altération de la note : 01 spécifie qu'une altération descendante (bémol ou double bémol) est associée à la note, 10 désigne la note non altérée et 11 spécifie qu'une altération montante (dièse ou double dièse) est associée à la note. La première partie d'un code de sol#[16] est identique à celle d'un la♭, leur deuxième partie étant respectivement 11 et 01. La hauteur correspondant à la note sol peut être représentée par un fa## (deuxième partie = 11), par un sol (deuxième partie = 10) ou par un la♭♭ (deuxième partie = 01). Ce codage très compact a cependant une

[15] Logi+ parle de code, nous éviterons cette appellation qui nous paraît, dans ce contexte, assez ambiguë.

[16] dans un souci de clarté, nous utilisons les abréviations ♭ pour bémol, ♭♭ pour double bémol, # pour dièse et ## pour double dièse.

limite. Rappelons que les notes mi, fa et si, do ne sont séparées que par un demi-ton. Le codage des altérations empêche de voir la hauteur du si♭ comme un do♭♭, celle du do# comme un si##. Cette limite est toutefois acceptée car ces notes, bien qu'acceptables en théorie, ne sont presque jamais utilisées.

La troisième partie (12 bits) du paquet « Note » détermine sa durée. L'unité de durée est le tick, qui équivaut à 1/96 noire. Cette division permet une précision de l'ordre de la sextuple croche.

2.1.2 Paquet « Silence »

paquet : `1000 0000 0000 dddd dddd dddd`

Outre son en-tête, le paquet « silence » ne contient qu'une partie déterminant sa durée. Son principe est similaire à la troisième partie d'un paquet « note ».

2.1.3 Paquet « Tonalité »

paquet : `1001 0000 0000 0000 0000 aaaa`

Les quatre derniers bits représentent un nombre signé déterminant le nombre d'altérations d'armature en vigueur à partir de ce code. 0 = rien à la clef ; 1, 2, ..., 7 = nombre de dièses à la clef ; -1, -2, ..., -7 = opposé du nombre de bémols à la clef.

2.1.4 Paquet « Clef »

paquet : `1001 1000 0000 0000 00tt 0ccc`

La première partie de ce paquet, d'une longueur de deux bits, représente le type de clef (00 = clef de sol, 10 = clef d'ut, 11 = clef de fa). La seconde partie (trois bits) désigne la ligne de référence de la clef.

2.1.5 Paquet « Mesure »

paquet : `1010 000r nnnn nnnn dddd dddd`

On observe ici deux parties principales : le codage du numérateur de la mesure sur le deuxième octet et du dénominateur sur le troisième octet. Ils peuvent donc varier tous deux de 0 à 255.

Le dernier bit du premier octet est positionné à 1 si la mesure doit apparaître sous forme résumée.

2.1.6 Paquet « Barre de mesure »

paquet : `1011 0000 0000 0000 0000 0000`

Ce paquet signale la présence d'une barre de mesure. Aucune information complémentaire n'y est associée.

2.1.7 Paquet « En-tête »

paquet : `1100 0mar tttt tttt tttt tttt`

Observons tout d'abord les trois derniers bits du premier octet. Le bit `m` renseigne sur le caractère contraignant des indications de mesure ; utile lors de l'édition des mélodies, phase où des vérifications doivent être réalisées, il est de moindre importance dans le problème qui nous occupe. Les bits `a` et `r` sont des indicateurs de la pertinence des données encodées. En effet, il existe des fiches encodées dans différentes versions de codage : où la tonalité du morceau n'est pas encodée (toutes les altérations sont accidentelles), où les durées de notes ne sont pas encodées. Le bit `r` positionné à 1 indique que les durées des notes sont arbitraires. Le bit `a` positionné à 1 indique que les altérations trouvées dans la mélodie ne se retrouvent peut-être pas telles quelles sur la partition originale.

Les deux derniers octets du paquet d'en-tête représentent le tempo. Ils indiquent la durée d'un tick en microsecondes. Il est à remarquer que des changements de tempo ne sont pas possibles dans le codage actuel vu qu'il n'existe qu'un paquet d'en-tête par mélodie (le premier paquet).

2.1.8 Paquet « n-olet »

paquet : `1101 0000 eeee nnnn 0000 llll`

Ce paquet est placé avant les paquets correspondant aux notes composant ce n-olet. Il comporte trois parties que nous appellerons `e`, `n` et `l` pour la clarté de l'explication. `l` représente le nombre de notes entrant dans le n-olet. `n` est le type de n-olet (n= 2 : duolet ; n = 3 : triolet...). `e` permet de calculer le ratio du n-olet. Comme nous l'avons remarqué au chapitre 1, les n-olets permettent de définir des fractions de durée de note : duolet de 2 notes qui s'exécutent sur la durée de 3 ; triolet de 3 notes qui s'exécutent sur la durée de 2, etc. En généralisant, un n-olet est composé de `l` notes dont la durée effective (entendue) est `n` / `e` * la durée dessinée (vue) de la note.

2.1.9 Paquet « Liaison »

paquet : `1110 0000 0000 nnnn 0000 0000`

Comme le paquet « n-olet », le paquet « liaison » est placé avant le paquet représentant la première note englobée dans cette liaison. La seule information accompagnant ce code est le nombre de notes incluses dans la liaison. Toutes les combinaisons de liaisons sont possibles (ex. : inclusion, chevauchement).

2.1.10 Paquet « Fin »

paquet : 1111 0000 0000 0000 0000 0000

Ce paquet marque la fin d'une mélodie.

2.2 Paramètres définis par l'utilisateur

L'utilisateur a le loisir de choisir la taille d'affichage des données musicales, leur position à l'écran, ainsi que la couleur de fond, la couleur des portées et des barres de mesures ainsi que la couleur des autres graphiques. Le choix de ces paramètres se fait dans un module de présentation qui n'entre pas dans le cadre de ce travail.

3. Contraintes non fonctionnelles

3.1 Langages de programmation

Depuis quelques années, Logi+ a adopté le langage C++ pour le développement de ses logiciels. Afin de pouvoir toucher une population d'utilisateurs la plus vaste possible, elle a décidé de concevoir des logiciels adaptés aux plates-formes couramment utilisées : Dos/Windows, MacIntosh et Unix. Le kit de portabilité d'xvt a été choisi pour faciliter cette compatibilité.

Ce Portability ToolKit consiste en une librairie de procédures C qui permettent de développer dans des environnements fenêtrés sans passer par des appels aux services de la plate-forme native. Selon l'environnement où le code est compilé, ces procédures seront traduites en différents codes natifs. Cela permet de construire des applications qui s'affichent de manière familière à l'utilisateur (avec les Objets Interactifs Concrets de Windows ou de MacIntosh, p.ex.) et qui fonctionnent à la vitesse d'un code natif sur différentes plates-formes, sans avoir à les recréer depuis le début[17].

L'afficheur musical a été transformé pour fonctionner sur le réseau Internet sous la forme d'une applet Java. Il s'agissait d'un projet-pilote pour Logi+ qui n'avait jamais utilisé ce nouveau langage de programmation. L'environnement de développement principal était ici le Java Development Kit (JDK) version 1.03 de Sun. Des tests ont été ensuite réalisés avec la version bêta du JDK 1.1.

3.2 Documentation

Au vu du nombre croissant de personnes impliquées dans le développement de logiciels à Logi+ (analystes-programmeurs, responsables de la maintenance, stagiaires...), des

[17] pour plus d'informations au sujet d'xvt et des solutions qu'il propose, voir http :// www.xvt.com

règles de documentation de programmes ont été établies. Elles ont pour but principal de faciliter la compréhension des algorithmes, d'en faciliter la maintenance, ... Dans cette section, nous énumérons les règles principales de documentation.

Tous les fichiers de code source commencent par un en-tête qui doit mentionner, au minimum, le contenu du fichier, le nom de son auteur et la date de sa création. Toute classe est définie dans un fichier d'extension **.h** et est implémentée dans un fichier de même nom mais d'extension **.cpp**. Dans le fichier de définition d'une classe, le programmeur commente toute variable et toute fonction membre. Ce commentaire explique le rôle de la variable ou de la fonction dans la classe. Dans le fichier d'implémentation d'une classe, un commentaire précède chaque fonction membre. Il renseigne sur le but de la fonction, son caractère public, privé ou protégé. Si le programmeur le juge utile, le commentaire peut être complété avec les pré- et post-conditions qui sont associées à la fonction. Toute instruction à l'intérieur de la fonction doit être commentée. Les commentaires associés à une instruction, à une définition de variable ou de fonction membre débutent à la colonne 49. Cela permet, si l'écran est suffisamment grand, une lecture parallèle du code et de la documentation.

Il est regrettable que la documentation des classes ne soit pas plus stricte. En effet, pour comprendre le but d'une classe (condition sine qua non de sa réutilisation), il est souvent nécessaire de descendre au niveau de l'implémentation des fonctions membres. Il est encore plus difficile de découvrir la durée de vie d'une classe. Ces faits occasionnent des pertes de temps non négligeables lors de l'intégration de sous-projets dans le projet principal, lors du choix d'une classe où encapsuler des données, ... Cette perte de temps est cependant acceptée par Logi+ vu la charge de travail déjà supportée par les programmeurs.

Chapitre 3. Conception

Dans ce chapitre, nous analysons le fonctionnement des différentes classes de l'afficheur musical.

Il est recommandé de lire ce chapitre avec, en parallèle, les graphiques synoptiques de l'annexe 1. Vous y trouvez un graphique montrant les relations de type « contient », un graphique montrant les relations d'utilisation (sous-traitance) et un dernier illustrant les relations d'héritage. La relation de type « contient » sous-entend une relation de responsabilité d'objet à objet. Si l'objet contient une simple référence vers un autre objet, sans en être responsable, la relation n'est pas mentionnée. Toutes les relations d'invocation d'objet à objet ne se retrouvent pas dans les relations de sous-traitance. Entre autres, les appels au constructeur ou à des fonctions d'accès ne sont pas mentionnés. Une relation de sous-traitance existe lorsque l'objet appelant utilise une méthode complexe de l'objet invoqué.

Dans un soucis de clarté, les fonctions d'accès ou de modification de variables membres ne sont pas reprises dans les tableaux ci-dessous, ces fonctions ne réalisant aucun traitement digne d'intérêt pour la compréhension des spécifications de la classe. De plus, les pré-conditions, lorsqu'elles ne concernent que le type des différents arguments de la méthode, ne sont pas mentionnées.

1. Classes d'interface

Dans cette section, nous analyserons les classes servant d'interface entre les classes du projet principal (gérant l'affichage de fiches) et les différents types de zones (texte, musique, dessin...). Les classes mères m'ont été données, tandis que la conception et l'implémentation des classes filles concernant des informations musicales entrent dans le cadre de mon projet.

Certaines méthodes des classes mères (telles Affiche ou Decoupe pour CZoneXVT) sont déclarées « virtuelles ». Les classes filles les redéfinissent afin de permettre le polymorphisme : l'afficheur appelle la méthode de la classe mère et celle qui est définie dans la classe fille est exécutée.

1.1 CZoneXVT

Données contenues	
Encadrement *encadrement*;	Type de cadre entourant la zone (aucun, normal ou arrondi).
DRAW_CTOOLS *outilsCadre*;	Paramètres d'affichage du cadre.
WINDOW *fenetre*;	Fenêtre où s'affiche la zone.
short *arrondiHorizontal*;	Taille de l'arrondi du cadre.
short *arrondiVertical*;	
long *contenuGauche*;	Position du contenu par rapport à la fenêtre.
long *contenuHaut*;	
long *contenuDroite*;	
long *contenuBas*;	
long *emplacementGauche*;	Position de la zone par rapport à la fenêtre.
long *emplacementHaut*;	
long *emplacementDroite*;	
long *emplacementBas*;	
Services offerts à tous	
CZoneXVT (WINDOW *fenetre*, const CbrouillonXVT **brouillon*)	Constructeur de l'objet. Outre la *fenetre* où s'affichera la zone, on indique un *brouillon* où sont stockées toutes les informations temporaires utiles.
void NoteDimensions (CBrouillonXVT **brouillon*, int *largeur*, int *hauteur*) const	{Post-condition} La *largeur* et la *hauteur* de la zone sont notées dans le *brouillon*.
void EmplacementAffichage (const CBrouillonXVT **brouillon*, long *&coinGauche*, long *&coinHaut*)	{Post-condition} *coinGauche* et *coinHaut* sont les coordonnées du point supérieur gauche de la zone par rapport à la fenêtre d'affichage.
BOOLEAN PrepareAffichage (const RCT **clipping*, long *coinGauche*, long *coinHaut*) const	{Pré-condition} *coinGauche* et *coinHaut* sont les coordonnées du point supérieur gauche de l'écran par rapport à la fenêtre. Les coordonnées du rectangle de clipping[18] sont dans *clipping*. {Post-condition} Renvoie TRUE si la zone et le rectangle de clipping se superposent. Le cadre entourant la zone est affiché. La zone de clipping de la fenêtre est fixée à l'intersection de *clipping* et du rectangle formé par la zone.

[18] on appelle rectangle de clipping la zone de l'écran qui doit être réaffichée (après le déplacement d'une fenêtre par exemple).

void CoinBasDroite (long *&coinDroite*, long *&coinBas*) const;	{Post-condition} *coinDroite* et *coinBas* sont les coordonnées du point inférieur droit de la zone par rapport à la fenêtre.
virtual BOOLEAN Decoupe (CBrouillonXVT **brouillon*) = 0;	{Pré-condition} *brouillon* est le brouillon associé à cette zone. {Post-condition} La zone est prête à être affichée.
virtual void Affiche (const RCT **clipping*, long *coinGauche*, long *coinHaut*) const = 0;	{Pré-condition} Le rectangle à afficher est contenu dans *clipping*. *coinGauche* et *coinHaut* sont les coordonnées du point supérieur gauche de l'écran par rapport à la fenêtre. {Post-condition} La zone est affichée si nécessaire.

A tous les types de zones doit correspondre une classe fille de CZoneXVT. Celle qui correspond à une zone contenant de l'information musicale est CZoneMusiqueXVT.

1.2 CZoneMusiqueXVT

Données contenues	
CVecteur *collection*; int *hauteurZone*; int *largeurZone*; CFenetreXVT **fenetreXVT*;	Ensemble des partitions (mélodies) de la zone. Hauteur de la zone. Largeur de la zone. Fenêtre où s'affiche la zone.
Services offerts à tous	
CZoneMusiqueXVT (WINDOW *fenetre*, CBrouillonMusiqueXVT **brouillon*);	Constructeur de l'objet. Outre la *fenetre* où s'affichera la zone, on indique un *brouillon* où sont stockées toutes les informations temporaires utiles.
~CZoneMusiqueXVT();	Destructeur de l'objet.

1.3 CBrouillonXVT

Données contenues	
BOOLEAN *finRangee*;	La zone fait-elle partie d'une colonne de fin de rangée ?
RCT *marges*;	Taille des différentes marges.
int *largeurZone*; int *hauteurZone*;	Largeur et hauteur de la zone.
CFenetreXVT **fenetreXVT*;	Fenêtre dans laquelle la zone associée au brouillon s'affichera.
const CZonePresentation **zonePresentation*;	Format de présentation de la zone associée.
const CMessagesZone **messagesZone*;	Liste des messages de la zone associée.
CZoneXVT **zoneXVT*;	Zone associée à ce brouillon.
int *largeurOptimale*;	La zone associée s'afficherait de manière optimale avec cette largeur.
int *largeurDisponible*;	Largeur disponible pour l'affichage de la zone associée.

Services offerts à tous	
CBrouillonXVT (CFenetreXVT **fenetre*, const CZonePresentation **zone*, const CMessagesZone **liste*);	Constructeur de l'objet. *fenetre* : fenêtre où la zone associée sera affichée. *zone* : format de présentation de la zone. *liste* : messages de la zone associée.
void LargeurDisponible (int *largeurUtilisee*);	{Post-condition} La variable membre *largeurDisponible* est calculée et initialisée en fonction de *largeurUtilisee*.
virtual void CalculeLargeurOptimale() = 0;	{Post-condition} La variable membre *largeurOptimale* est calculée et initialisée.
virtual CZoneXVT *NouvelleZoneXVT() = 0;	{Post-condition} La zone d'affichage définitive associée à ce brouillon est allouée et son adresse est donnée en résultat.

A tous les types de zones doit correspondre une classe fille de CBrouillonXVT. Celle qui correspond à une zone contenant de l'information musicale est CBrouillonMusiqueXVT.

1.4 CBrouillonMusiqueXVT

Cette classe ne définit pas de nouvelles méthodes ou variables membres. Elle redéfinit, pour la musique, les méthodes CalculeLargeurOptimale et NouvelleZoneXVT du CBrouillonXVT.

2. Classes graphiques

2.1 CGraphiqueMusical

Données contenues	
long *positionX* long *positionY*	Coordonnées de l'objet par rapport au coin supérieur gauche de la fenêtre.
Services offerts aux classes filles	
void AfficheImage (int *numero*, int *l*, int *h*, short *decalageX*, short *decalageY*, long *decalageGauche*, long *decalageHaut*, WINDOW *fenetre*, const CPartition **partition*) const	Affichage d'un composant de l'objet de type image. {Pré-condition} *decalageGauche* et *decalageHaut* représentent respectivement le décalage horizontal et vertical de la fenêtre *fenetre* par rapport à l'écran. La *partition* contient l'objet graphique. {Post-condition} L'image identifiée par *numero*, de largeur *l* et de hauteur *h*, s'affichera dans la fenêtre *fenetre* aux coordonnées du graphique décalées horizontalement de *decalageX* et verticalement de *decalageY*.

void AfficheLigne (const PNT **pointDebut*, const PNT **pointFin*, CPEN **stylo*, long *decalageGauche*, long *decalageHaut*, WINDOW *fenetre*, const CPartition **partition*) const	Affichage d'un composant de l'objet de type ligne. {Pré-condition} idem qu'AfficheImage. {Post-condition} Une ligne a été tracée du *pointDebut* au *pointFin* (coordonnées relatives à l'objet graphique) dans la *fenetre*. Cette ligne possède les caractéristiques de *stylo* (couleur, épaisseur, style de trait).
void AfficheNombre (short *n*, const XVT_FNTID *police*, short *translationX*, short *translationY*, long *decalageGauche*, long *decalageHaut*, WINDOW *fenetre*, const CPartition **partition*) const	Affichage d'un composant de l'objet de type nombre. {Pré-condition} idem qu'AfficheImage {Post-condition} Le nombre *n* a été affiché dans la *fenetre* aux coordonnées de l'objet décalées horizontalement de *translationX* et verticalement de *translationY*. La police de caractère utilisée est *police*.
Services offerts à tous	
CGraphiqueMusical (long *x*, long *y*);	Constructeur de l'objet. *x* et *y* sont respectivement ses coordonnées horizontale et verticale par rapport au coin supérieur gauche de la fenêtre.
virtual void Affiche (long *decalageGauche*, long *decalageDroit*, const CPartition **partition*, WINDOW *fenetre*) const;	Demande à l'objet graphique de s'afficher. {Pré-condition} idem qu'AfficheImage. {Post-condition} L'objet musical est affiché dans la *fenetre*.
virtual BOOLEAN Note() const;	Demande à l'objet s'il représente une note.
virtual BOOLEAN Silence() const;	Demande à l'objet s'il représente un silence.
virtual BOOLEAN GraphiqueFinMesure () const;	Demande à l'objet s'il représente un signe pouvant clôturer une mesure.
virtual int DecalageIdeal() const;	Demande à l'objet dans quelle proportion il peut être espacé de celui qui se trouve à sa droite.

Les classes CGraphiqueDiese, CGraphiqueBemol, CGraphiqueBecarre, CGraphiqueDoubleDiese, CGraphiqueDoubleBemol, CGraphiqueCercleRouge, CGraphiqueIconeMesure, CGraphiqueIconeRythmes, CGraphiqueIconeArmatures, CGraphiqueFond, CGraphiqueBarreDeNote, CGraphiqueLiaison, CGraphiqueNOlet, CGraphiqueCleSol, CGraphiqueCleFa, CGraphiqueCleUt, CGraphiqueMesure, CGraphiqueMesureC, CGraphiqueMesureCBarre, CGraphiqueMesureResumee, CGraphiquePortee, CGraphiqueTempo, CGraphiquePointe, CGraphiqueContinuateurDeMesure et CGraphiqueBarreDeMesure héritent de la classe CGraphiqueMusical. Les classes finales qui n'incluent pas plus d'informations et qui n'offrent pas plus de services que leur classe mère

(c'est-à-dire celles qui ne font que redéfinir des fonctions déjà expliquées) ne seront pas analysées par la suite. Il en sera ainsi dans toutes les sections suivantes.

2.2 CGraphiqueFond

Données contenues	
short *largeur*; short *hauteur*;	Largeur et hauteur du fond.
Services offerts à tous	
CGraphiqueFond (long *x*, long *y*, short *l*, short *h*);	Constructeur de l'objet. *x* et *y* sont les coordonnées de l'objet graphique. *l* est la largeur du fond et *h* sa hauteur.

2.3 CGraphiqueLiaison

Données contenues	
RCT *ovale*; PNT *pointDebut*; PNT *pointFin*;	Le graphique de la liaison est un arc de l'ovale tangent aux milieux des côtés du rectangle *ovale* ; cet arc commence au *pointDebut* et se termine au *pointFin*.
Services offerts à tous	
CGraphiqueLiaison (long *x*, long *y*, RCT **ovale*, PNT **pointDebut*, PNT **pointFin*);	Constructeur de l'objet. *x* et *y* sont les coordonnées de l'objet ; les autres paramètres correspondent aux variables expliquées ci-dessus.

2.4 CGraphiqueNOlet

Représente un crochet annonçant un n-olet.

Données contenues	
PNT *pointDebut*; PNT *pointFin*; BOOLEAN *orientationBasse*; short *n*;	Coordonnées des coins du crochet. Les pattes du crochet sont orientées vers le bas si *orientationBasse* = TRUE. Chiffre apparaissant dans le crochet.
Services offerts à tous	
CGraphiqueNOlet (long *x*, long *y*, PNT **pointDebut*, PNT **pointFin*, short *n*, BOOLEAN *orientationBasse*);	Constructeur de l'objet. *x* et *y* sont les coordonnées de l'objet ; les autres paramètres correspondent aux variables membres expliquées ci-dessus.

2.5 CGraphiqueBarreDeNote

Données contenues	
PNT *pointDebut*; PNT *pointFin*;	Points de début et de fin de la barre reliant un groupe de notes.

Services offerts à tous	
CGraphiqueBarreDeNote (long *x*, long *y*, PNT **pointDebut*, PNT **pointFin*);	Constructeur de l'objet. *x* et *y* sont les coordonnées de l'objet ; les autres paramètres correspondent aux variables expliquées ci-dessus.

2.6 CGraphiqueMesure

Données contenues	
int *numerateur*; int *denominateur*;	Numérateur et dénominateur de la mesure représentée par l'objet.
Services offerts à tous	
CgraphiqueMesure (int *numerateur*, int *denominateur*, long *x*, long *y*);	Constructeur de l'objet. *x* et *y* sont les coordonnées de l'objet ; les autres paramètres correspondent aux variables expliquées ci-dessus.

2.7 CGraphiqueMesureResumee

Données contenues	
int *numerateur*; int *denominateur*;	Numérateur et dénominateur de la mesure représentée par cet objet.
Services offerts à tous	
CGraphiqueMesureResumee (int *numerateur*, int *denominateur*, long *x*, long *y*);	Constructeur de l'objet. *x* et *y* sont les coordonnées de l'objet ; les autres paramètres correspondent aux variables expliquées ci-dessus.

2.8 CGraphiqueTempo

Données contenues	
int *tempo*;	Nombre de noires à la minute (tempo)
Services offerts à tous	
CGraphiqueTempo (int *tempo*, long *x*, long *y*);	Constructeur de l'objet. *x* et *y* sont les coordonnées de l'objet ; *tempo* est le nombre de noires à la minute.

2.9 CGraphiquePointe

Données contenues	
short *nombrePoints*;	Nombre de points relatifs par cet objet.

Services offerts aux classes filles	
AffichePoints (short *decalageHorizontal*, short *milieuGraphique*, long *decalageGauche*, long *decalageHaut*, WINDOW *fenetre*, const CPartition **partition*) const	Affichage des points de l'objet. {Pré-condition} *decalageGauche* et *decalageHaut* représentent respectivement le décalage horizontal et vertical de la fenêtre *fenetre* par rapport à l'écran. La *partition* contient l'objet graphique. *DecalageHorizontal* indique l'abscisse (par rapport à l'objet) à partir de laquelle des points peuvent être affichés. {Post-condition} Les points du graphiques ont été affichés à droite de l'objet fille à l'ordonnée *milieuGraphique* - 2 (par rapport à l'objet).
Services offerts à tous	
CGraphiquePointe (long *x*, long *y*, short *nombrePoints*);	Constructeur de l'objet. *x* et *y* sont les coordonnées de l'objet ; *nombrePoints* est le nombre de points de l'objet.

Les classes CGraphiqueNote et CGraphiqueSilence héritent de CGraphiquePointe.

2.10 CGraphiqueNote

Données contenues	
short *nombreLignes*;	Nombre de barres parallèles à la portée ajoutées pour cette note (cfr figure 2).
int *duree*;	Durée de la note en 128ème de noire.
const CGraphiquePortee **portee*;	Portée dans laquelle la note est incluse.
Services offerts aux classes filles	
BOOLEAN AfficheLignes (short *largeur*, long *decalageGauche*, long *decalageHaut*, WINDOW *fenetre*, const CPartition **partition*) const	Affichage des barres parallèles à la portée. {Pré-condition} *decalageGauche* et *decalageHaut* représentent respectivement le décalage horizontal et vertical de la fenêtre *fenetre* par rapport à l'écran. La *partition* contient l'objet graphique. {Post-condition} Des barres (de largeur *largeur*) sont affichées entre la portée et la note.
Services offerts à tous	
CGraphiqueNote (long *x*, long *y*, short *nombreLignes*, int *duree*, short *nombrePoints*);	Constructeur de l'objet. *x* et *y* sont les coordonnées de la note ; *nombrePoints* est le nombre de points de la note ; *durée* et *nombreLignes* correspondent aux variables décrites ci-dessus.

Les classes CGraphiqueCarree, CGraphiqueRonde et CGraphiqueNoteHampe héritent de CGraphiqueNote.

2.11 CGraphiqueSilence

Données contenues	
int *duree*;	Durée du silence en 128ème de noire.
Services offerts à tous	
CGraphiqueSilence (long *x*, long *y*, int *duree*, short *nombrePoints*);	Constructeur de l'objet. *x* et *y* sont les coordonnées du silence ; *nombrePoints* est le nombre de points du CGraphiquePointe ; *duree* correspond à la variable *duree* décrite ci-dessus.

Les classes CGraphiqueDoublePause, CGraphiquePause, CGraphiqueDemiPause, CGraphiqueSoupir, CGraphiqueDemiSoupir, CGraphiqueQuartSoupir, CGraphique8iemeSoupir, CGraphique16iemeSoupir, CGraphique32iemeSoupir et CGraphique64iemeSoupir héritent de CGraphiqueSilence.

2.12 CGraphiqueNoteHampe

Données contenues	
short *hauteurHampe*; BOOLEAN *hampeHaute*;	Hauteur de la hampe en pixels(cfr figure). Indique l'orientation de la hampe.
Services offerts aux classes filles	
void AfficheHampe (short *arrondiNote*, short *hauteurNote*, short *largeurNote*, long *decalageGauche*, long *decalageHaut*, WINDOW *fenetre*, const CPartition **partition*) const;	Affichage des points de l'objet. {Pré-condition} *decalageGauche* et *decalageHaut* représentent respectivement le décalage horizontal et vertical de la fenêtre *fenetre* par rapport à l'écran. La *partition* contient l'objet graphique. *arrondiNote*, *hauteurNote* et *largeurNote* : cfr figure. {Post-condition} La hampe de la note a été affichée.
Services offerts à tous	
CGraphiqueNoteHampe (long *x*, long *y*, short *hauteur*, BOOLEAN *orientation*, short *nombreLignes*, int *duree*, short *nombrePoints*);	Constructeur de l'objet. *x* et *y* sont les coordonnées de l'objet ; *nombrePoints*, *nombreLignes* et *duree* correspondent aux variables des classes supérieures. *hauteur* représente la hauteur de la hampe et *orientation* = TRUE si la hampe est orientée vers le haut.

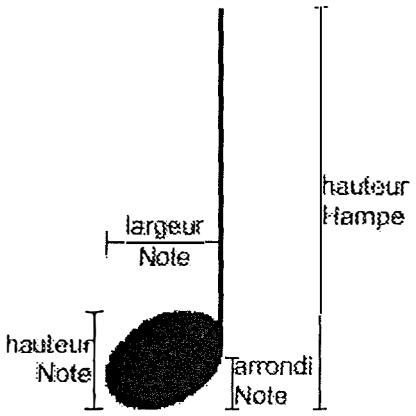


Figure 15: CGraphiqueNoteHampe

Les classes CGraphiqueBlanche et CGraphiqueNoteMoustachue héritent de CGraphiqueNoteHampe.

2.13 CGraphiqueNoteMoustachue

Données contenues	
short *nombreMoustachesGraphiques*; short *nombreMoustachesLogiques*;	Nombre de moustaches affichées de cette note. Nombre de moustaches non affichées de cette note (quand la note fait partie d'un groupe de notes reliées par une ou des barres).
Services offerts aux classes filles	
void AfficheMoustaches (long *decalageGauche*, long *decalageHaut*, WINDOW *fenetre*, const CPartition **partition*) const;	Affichage des moustaches de la note. {Pré-condition} *decalageGauche* et *decalageHaut* représentent respectivement le décalage horizontal et vertical de la fenêtre *fenetre* par rapport à l'écran. La *partition* contient l'objet graphique. {Post-condition} Les moustaches sont affichées correctement par rapport à la note.
Services offerts à tous	
CGraphiqueNoteMoustachue (long *x*, long *y*, short *hauteurHampe*, BOOLEAN *hampeHaute*, short *nombreMoustaches*, short *nombreLignes*, int *duree*, short *nombrePoints*);	Constructeur de l'objet. Outre les paramètres concernant les variables héritées, *nombreMoustaches* est le nombre de moustaches non affichées. A la création de l'objet, le nombre de moustaches affichées est nul.

void AjouteMoustachesGraphiques (float *echelle*);	{Pré-condition} *echelle* est l'échelle d'affichage de la musique {Post-condition} Le nombre de moustaches affichées prend la valeur du nombre de moustaches non affichées.

La classe CGraphiqueNoire hérite de CGraphiqueNoteMoustachue.

3. Classes relatives au cache de pixmaps

L'affichage des images composant les graphiques musicaux se fait par l'intermédiaire d'un cache. Les images sont stockées en noir sur fond blanc dans un fichier de ressources. Chaque affichage demande un traitement de changement de couleur et de mise à l'échelle de l'image. Afin d'améliorer les performances, et donc de limiter l'effet de clignotement lors de l'affichage, le cache garde en mémoire les dernières images utilisées. Ces images réduites à une certaine échelle sont stockées en double : en couleur et en noir-et-blanc.

3.1 CCachePixmap

Données contenues	
CVecteur *cache*; int *taille*;	Liste des pixmaps stockées dans le cache. Taille du cache.
Services offerts à tous	
CCachePixmap(int *taille*);	Constructeur du cache. Le *taille* du cache est spécifiée en paramètre.
~CCachePixmap()	Destructeur du cache {Post-condition} Le cache et tous ses éléments sont détruits.
XVT_PIXMAP Pixmap (int *numeroRessourceDessin*, COLOR *couleur*, float *echelle*, short *largeur*, short *hauteur*, BOOLEAN *enCouleur*)	{Pré-condition} *numeroRessourceDessin* est un identifiant valide d'un dessin. *couleur*, *largeur* et *hauteur* sont la couleur, la largeur et la hauteur du dessin demandé. {Post-condition} Si la pré-condition n'est pas vérifiée, la constante NULL_PIXMAP est renvoyée. Sinon, la pixmap correspondant à l'identifiant *numeroRessourceDessin* est renvoyée en noir-et-blanc si *enCouleur* = FALSE ; en couleur sinon.
void VideCache()	{Post-condition} Les éléments du cache utilisés le moins récemment sont supprimés pour que le nombre d'éléments du *cache* = *taille*.

3.2 CElementPixmap

Données contenues	
int *numeroRessource*;	Numéro de ressource identifiant le dessin de l'élement.
COLOR *couleur*;	Couleur de la pixmap couleur de l'élément.
float *echelle*;	Echelle de redimensionnement des dessins utilisée pour cet élément.
XVT_PIXMAP *pixmapNoire*;	Pixmap noir sur fond blanc stocké.
XVT_PIXMAP *pixmapCouleur*;	Pixmap couleur sur fond blanc stocké.
Services offerts à tous	
CElementCachePixmap (COLOR *couleur*, float *echelle*);	Constructeur de l'élément de cache. Les variables membres *couleur* et *echelle* sont initialisées.
~CElementCachePixmap()	Destructeur de l'élément de cache
BOOLEAN Initialise (int *numeroRessourceDessin*, short *largeur*, short *hauteur*)	{Pré-condition} *numeroRessourceDessin* est l'identifiant valide d'un dessin. *largeur* et *hauteur* sont la largeur et la hauteur du dessin demandé. {Post-condition} Le résultat est TRUE et les pixmaps membres de l'élément sont correctement initialisés ou le résultat est FALSE.

4. Classe CPartition

Une zone de données musicales, on l'a vu plus haut, peut comporter plusieurs mélodies. Chacune de ces mélodies est gérée par un objet de type CPartition.

Données contenues	
const CMessagePresentation **melodie*;	Code de la mélodie gérée par cette partition.
const CZoneMusiqueXVT **zone*;	Zone dans laquelle cette partition s'affiche.
DRAW_CTOOLS *outils*;	Outils utilisés pour dessiner les objets de cette partition.
float *echelle*;	Echelle de redimensionnement des graphiques utilisée pour l'affichage de cette partition.
int *hauteur*;	Hauteur de la partition en pixels.
CVecteur *partition*;	Liste des objets graphiques composant cette partition.
Services offerts à tous	
CPartition (const CMessagePresentation * *melodie*, const CZoneMusiqueXVT **zone*)	Constructeur de la partition. Les variables membres *melodie* et *zone* sont initialisées.
~CPartition()	Destructeur de la partition {Post-condition} La partition ainsi que tous les objets graphiques qui la composent sont détruits.

BOOLEAN Decoupe (int *largeurDisponible*, long *margeGauche*, long *margeHaut*)	{Post-condition} Les objets graphiques de la partition (*partition*) sont initialisés en fonction de *melodie*. Les objets sont organisés sur toute la *largeurDisponible* (la gestion des marges dans une zone musicale n'est pas encore implémentée). Le résultat est TRUE si le découpage a réussi.
void Affiche (long *coinGauche*, long *coinHaut*, WINDOW *fenetre*) const	{Pré-condition} *coinGauche* et *coinHaut* représentent les coordonnées du coin supérieur gauche de l'écran par rapport à la *fenetre*. {Post-condition} Les objets graphiques de la partition sont affichés dans la *fenetre*.
void TranslateVerticalement (int *translation*);	{Post-condition} Tous les objets graphiques de la partition ont été translatés de *translation* pixels vers le bas.
inline WINDOW Fenetre() const;	Accès à la fenêtre où s'affiche la partition.
inline CCachePixmap *CachePixmap() const;	Accès au cache de pixmaps utilisé lors de l'affichage des objets graphiques.

5. Classes de découpage

5.1 CDecoupageMusical

Données contenues	
long *positionX*; long *positionY*; int *largeur*;	Coordonnées de l'objet dans la fenêtre. Largeur de l'objet.
Services offerts à tous	
CDecoupageMusical (long *abscisse*, long *ordonnee*, int *largeurDesiree*)	Constructeur de l'objet de découpage. Les variables membres sont initialisées.

Les classes CDecoupagePartition, CDecoupagePortee et CDecoupageMesure héritent de cette classe.

5.2 CDecoupagePartition

Les objets dérivés de cette classe supervisent le découpage au niveau de la partition (c'est-à-dire au niveau d'une mélodie).

Données contenues	
CPartition **partition*; short *ton*; CMusiqueMesure *mesure*; CMusiqueCle *cle*; int *tempo*; int *largeurDisponible*; int *hauteurPartition*; CVecteur *brouillonLiaisons*; CVecteur *portees*;	Partition associée à cet objet. Tonalité à l'endroit courant du découpage. Mesure à l'endroit courant du découpage. Clef à l'endroit courant du découpage. Tempo à l'endroit courant du découpage. Largeur disponible pour l'affichage de la partition. Hauteur de la partition. Repères utiles au placement des liaisons. Liste des portées (de découpage) gérées par cette partition.
Services offerts à tous	
CDecoupagePartition (CPartition **partition*, int *largeurOctroyee*, long *abscisse*, long *ordonnee*)	Constructeur de l'objet de découpage. *partition* correspond à la variable membre *partition*, *largeurOctroyée* à *largeurDisponible*. Les variables *ton*, *mesure* et *cle* sont initialisées respectivement à 0, 4/4 non résumée et sol 2ème ligne.
~CDecoupagePartition()	Destructeur de l'objet de découpage. {Post-condition} L'objet, ainsi que les brouillons de liaisons et les portées de découpage qu'il contient, sont détruits.
BOOLEAN Ajoute (const unsigned char **&melodie*, int *decalage*)	{Pré-condition} *decalage* est le décalage vertical que doivent subir les objets musicaux relatifs à la partition courante pour apparaître correctement dans la zone (sous d'autres partitions p.ex.). {Post-condition} Les objets graphiques correspondant aux codes musicaux de *melodie* sont ajoutés à la zone. Si un problème est survenu lors du découpage, on renvoie FALSE.
inline float Echelle() const;	Accès à l'échelle de redimensionnement des objets graphiques de la partition.
inline CVecteur *ObjetsGraphiques();	Accès à la liste des objets graphiques de la partition.

5.3 CDecoupagePortee

Les objets dérivés de cette classe supervisent le découpage au niveau d'une portée de la partition.

Données contenues	
CDecoupagePartition **partition*; short *depassementHaut*; short *depassementBas*; CGraphiquePortee **saPortee*; CVecteur *mesures*;	Partition (de découpage) contenant cet objet cfr figure 16. Référence de la portée graphique associée à cette portée de découpage. Liste des mesures (de découpage) gérées par cette portée.
Services offerts à tous	
CDecoupagePortee (CDecoupagePartition **partition*, long *abscisse*, long *ordonnee*)	Constructeur de l'objet de découpage. *partition* correspond à la variable membre *partition*. Les deux autres variables sont les coordonnées de l'objet.
~CDecoupagePortee()	Destructeur de l'objet de découpage. {Post-condition} L'objet ainsi que les mesures de découpage qu'il contient sont détruits.
void ChangeDepassements()	{Post-condition} $depassementHaut = \max \{depassementHaut_0 \cup \{depassementHaut_m\}\}$, $\forall$ m $\in$ mesures de découpage contenues. $depassementBas = \min \{depassementBas_0 \cup \{depassementBas_m\}\}$, $\forall$ m $\in$ mesures de découpage contenues.
BOOLEAN Ajoute (const unsigned char **&melodie*);	{Pré-condition} *melodie* est le premier code concernant cette portée. {Post-condition} La partition (graphique) contient une portée (graphique) de plus et les graphiques musicaux contenus (musicalement parlant) dans cette portée sont ajoutés à la partition. *melodie* référence le premier code de la portée suivante. On renvoie FALSE si on n'a pas pu découper une portée, entre autre cas si la largeur disponible ne permet pas de traiter un seul code de la mélodie.
void Justifie();	{Post-condition} Les objets graphiques ont été écartés pour qu'ils occupent toute la largeur disponible de la portée.
void TranslationVerticale(long *decalage*)	{Post-condition} Les objets graphiques de la portée ont été décalés de *decalage* pixels vers le bas.

BOOLEAN AjouteBarresNotes()	{Post-condition} Les repères utiles à la création de graphiques de barres de notes de cette portée ont été transformés en barres de notes.
BOOLEAN AjouteNOlets()	{Post-condition} Les repères utiles à la création de graphiques de n-olets de cette portée ont été transformés en graphiques de n-olets.
BOOLEAN TraiteLiaison (CVecteur **graphiques*, CBrouillonGraphiqueLiaison **liaison*, BOOLEAN *&liaisonTraitee*)	{Pré-condition} *graphiques* est l'ensemble des objets graphiques de la partition. *liaison* est un repère utile à la création d'un graphique de liaison de la partition. {Post-condition} La sous-liaison (CGraphiqueLiaison) relative à la *liaison* et à la portée a été ajoutée aux objets *graphiques*. *liaisonTraitee* = TRUE si la *liaison* se termine sur cette portée. On renvoie FALSE si une erreur s'est produite lors de traitement de cette liaison.
inline float Echelle() const;	Accès à l'échelle de redimensionnement des objets graphiques de la partition.
inline int LargeurDisponible() const;	Calcul de la largeur disponible dans la portée.
inline int NombreMesures();	Calcul du nombre de mesures (de découpage) contenues dans la portée.
inline CDecoupageMesure * PremiereMesure() const;	Accès à la première mesure (de découpage) de la portée.
inline CDecoupageMesure * DerniereMesure() const;	Accès à la dernière mesure (de découpage) de la portée.

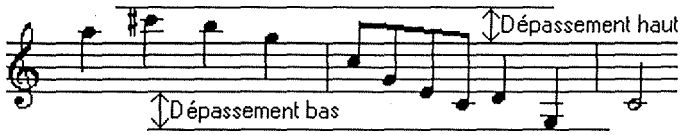


Figure 16: dépassements

5.4 CDecoupageMesure

Les objets dérivés de cette classe supervisent le découpage au niveau d'une mesure d'une portée de la partition.

Données contenues	
CDecoupagePartition **partition*;	Partition dans laquelle se trouve la mesure.
CDecoupagePortee **portee*;	Portée dans laquelle se trouve la mesure.
short *depassementHaut*;	cfr figure 16.
short *depassementBas*;	
short *nombreNotesNOlet*;	Nombre de notes du n-olet courant déjà traitées (= 0 si on ne se trouve pas dans un n-olet).
int *positionDansPortee*;	Position du début de la mesure (point extérieur gauche) par rapport au début de la portée.
short *alterationsCourantes*[7];	Altérations subies par les notes de la mesure à un instant donné.
CVecteur *brouillonNOlets*;	Ensemble des repères utiles à la création des objets graphiques de n-olets de la mesure.
CVecteur *brouillonBarresNotes*;	Ensemble des repères utiles à la création des barres reliant des groupes de notes de la mesure.
CVecteur *graphiques*;	Ensemble des objets graphiques contenus dans la mesure.
Services offerts à tous	
CDecoupageMesure (CDecoupagePartition **partition*, CDecoupagePortee **portee*, long *abscisse*, long *ordonnee*);	Constructeur de l'objet de découpage. *partition* et *portee* correspondent respectivement aux variables membres *partition* et *portee*. Les deux autres variables sont les coordonnées de l'objet. {Post-condition} Les *alterationsCourantes* sont initialisées selon le ton courant de la partition. depassementHaut = depassementBas = nombreNotesNOlet = 0.
~CDecoupageMesure();	Destructeur de l'objet de découpage. {Post-condition} L'objet ainsi que les repères utiles à la création de graphiques qu'il contient sont détruits.
int IndicePremiereNote() const;	{Post-condition} Renvoie l'indice de la première note de la mesure dans le tableau des objets graphiques de la partition. S'il n'existe pas de notes dans la mesure, renvoie 0.
int IndiceDerniereNote() const;	{Post-condition} Renvoie l'indice de la dernière note de la mesure dans le tableau des objets graphiques de la partition. S'il n'existe pas de notes dans la mesure, renvoie 0.

BOOLEAN Ajoute (const unsigned char *&*melodie*, BOOLEAN &*mesureComplete*);	{Pré-condition} *melodie* est le premier code concernant cette mesure. {Post-condition} On a essayé d'inclure un maximum de objets graphiques dans la mesure (et en parallèle dans la partition). Plusieurs cas sont possibles : explications dans le tableau suivant. On renvoie FALSE si une erreur s'est produite lors d'un ajout d'objet graphique.
void TranslationVerticale(long *decalage*);	{Post-condition} Les objets graphiques de la mesure ont été décalés de *decalage* pixels vers le bas.
void ArrangeGraphiques (short *decalage*, short *pixelsSupplementaires*);	{Post-condition} Tous les objets graphiques de la mesure ont subi un décalage de *decalage* pixels vers la droite. Ils ont, en outre, subi un autre décalage afin que la mesure soit *pixelsSupplementaires* pixels plus large. Ce décalage est calculé en fonction des *decalageIdeal* de chaque objet graphique.
void EffaceObjetsGraphiques();	{Pré-condition} Cette mesure est la dernière qui ait été ajoutée à la partition. {Post-condition} Les objets graphiques de cette mesure ont été détruits et supprimés de l'ensemble des objets graphiques de la partition.
BOOLEAN AjouteContinuateur();	{Post-condition} Un objet graphique de continuateur de mesure a été ajouté à droite de tous les objets graphiques de la mesure. On renvoie TRUE si tout s'est bien passé.
BOOLEAN AjouteArmature (const unsigned char *& *codeATraiter*);	{Pré-condition} *codeATraiter* référence le premier code non encore traité. {Post-condition} Une clef et des altérations d'armature ont été ajoutées à la mesure en fonction de la clef et du ton courant de la partition. On renvoie TRUE si tout s'est bien passé.
BOOLEAN AjouteBarresNotes()	{Post-condition} Les objets graphiques de barre groupant des notes ont été ajoutés en fonction des repères existants. On renvoie TRUE si tout s'est bien passé.

BOOLEAN AjouteNOlets()	{Post-condition} Les objets graphiques de n-olets ont été ajoutés en fonction des repères existants. On renvoie TRUE si tout s'est bien passé.

Partie de la post-condition de la méthode Ajoute :

Mesure Complète	Mesure non complète	
	une seule mesure dans la portée	plusieurs mesures dans la portée
mesureComplete = TRUE	mesureComplete = FALSE	
les repères préalables à la création des objets graphiques de n-olets et de barres groupant des notes ont été créés		Les repères concernant les barres groupant les notes ne sont pas créés
melodie référence le premier code à traiter dans la mesure suivante		*melodie* inchangée

6. Classes musicales

Ces classes d'objets représentent des concepts purement musicaux et le comportement qui leur est associé.

6.1 CMusiqueCle

Données contenues	
nomCle *nom*; int *ligne*; short **hauteurAlterations*;	Nom de la clef : cleSol, cleUt ou cleFa. Ligne sur laquelle la clef est dessinée. Hauteurs des altérations d'armature dans cette clef.
Services offerts à tous	
CMusiqueCle (nomCle *nom*, int *ligne*);	Constructeur de l'objet clef. Les arguments sont le nom de la clef et la ligne sur laquelle la clef se dessine. {Post-condition} Les hauteurs des altérations dans cette clef ont été initialisées.

6.2 CMusiqueMesure

Données contenues	
short *numerateur*; short *denominateur*; BOOLEAN *resumee*;	Numérateur de la mesure. Dénominateur de la mesure. La mesure est-elle résumée ?
Services offerts à tous	
CMusiqueMesure (short *numerateur*, short *denominateur*, BOOLEAN *resumee*)	Constructeur de l'objet mesure. Les arguments sont le numérateur, le dénominateur de la mesure et le fait que la mesure est représentée sous forme résumée ou pas.

6.3 CMusiqueHauteurNote

Données contenues	
short *octave*; nom *hauteur*;	Numéro de l'octave de la note. Nom de la note (do, re, mi, fa, sol, la ou si).
Services offerts à tou	
enum nom {Do = 0, Re, Mi, Fa, Sol, La, Si};	Suite ordonnée de noms possibles de note.
CMusiqueHauteurNote (short *octave*, nom *hauteur*)	Constructeur de l'objet hauteur de note. Les arguments sont l'octave et le nom de la note.
void Transpose(short *ecart*);	{Post-condition} La hauteur de la note a été augmentée de *ecart* notes.
void TransposeCleSol2(CMusiqueCle **cle*);	{Pré-condition} Objet = $this_0$. {Post-condition} Objet = $this_1$. Si $this_0$ représente une hauteur de note dont le graphique se dessine à une hauteur *h* dans la portée avec la clef *cle*, $this_1$ représente la hauteur d'une note affichée à la même hauteur *h* dans une portée avec la clef de sol 2ème ligne.
int CalculeLignesSupplementaires();	{Post-condition} On renvoie le nombre de lignes de portée supplémentaires (cfr figure 2) que nécessite la hauteur de la note (supposée être exprimée en fonction de la clef de sol 2ème ligne).
int CalculeOrdonnee() const;	{Post-condition} On renvoie l'ordonnée du corps de la note relatif à la hauteur (supposée être exprimée en fonction de la clef de sol 2ème ligne).

6.4 CMusiqueNote

Données contenues	
CMusiqueHauteurNote *hauteur*; alterateur *alteration*; int *duree*; int *rythmeVu*;	Hauteur de la note. Altération subie par la note. Durée (exprimée en ticks) de la note. Durée vue (exprimée en ticks) de la note : une noire faisant partie d'un n-olet aura une durée vue égale à celle d'une noire normale.
Services offerts à tous	
enum alterateur {DoubleBemol = -2, Bemol, Becarre, Diese, DoubleDiese};	Suite ordonnée des altérations possibles d'une note.

CMusiqueNote (const CMusiqueHauteurNote **hauteur*, alterateur *alteration*, int *duree*, int *rythme*) ;	Constructeur de l'objet note. Les arguments sont la hauteur de la note, son altération, sa durée et sa durée vue.

7. Classes d'initialisations de graphiques

7.1 CInitialisateurGraphique

Données contenues	
int *ordonnee* ;	Ordonnée de l'objet graphique par rapport à la première ligne de la portée.
short *depassementHaut* ; short *depassementBas* ;	Dépassements du graphique en haut et en bas de la portée (cfr figure 16).
short *largeurGraphique* ;	Largeur du graphique en pixels.
short *espaceAvant* ; short *espaceApres* ;	Espace en pixels devant être laissé avant et après le graphique.
CGraphiqueMusical* *graphique* ;	Objet graphique initialisé.
Services offerts à tous	
CInitialisateurGraphique(int *ordonnee*) ;	Constructeur de l'objet initialisateur. {Post-condition} L'ordonnée du graphique est *ordonnee*, tous les autres paramètres sont nuls.
BOOLEAN Initialise (float *echelle*, long *positionX*, int *largeur*) = 0 ;	{Pré-condition} *echelle* est l'échelle de redimensionnement des objets graphiques. *positionX* et *largeur* sont respectivement l'abscisse et la largeur courante de la mesure de découpage dans laquelle l'objet va être inclu. {Post-condition} Les variables membres sont initialisées, on renvoie TRUE si tout c'est bien passé.

Les classes CInitialisateurAlterationGraphique et CInitialisateurGraphiqueRythme héritent de cette classe.

7.2 CInitialisateurAlterationGraphique

Données contenues	
CMusiqueNote : :alterateur *alteration* ;	Altération que représente le graphique.
Services offerts à tous	
CInitialisateurAlterationGraphique (int *ordonnee*, CMusiqueNote : :alterateur *alteration*) ;	Constructeur de l'objet initialisateur. {Post-condition} L'ordonnée du graphique est *ordonnee* et l'altération représentée est *alteration*.

7.3 CInitialisateurGraphiqueRythme

Classe d'initialisation d'un objet graphique ayant une notion de durée comme paramètre.

Données contenues	
int *rythme* ; int *duree* ;	Durée vue de la note (en ticks). Durée réelle de la note (en ticks).
Services offerts à tous	
CInitialisateurGraphiqueRythme (int *ordonnee*, int *duree*, int *rythme*)	Constructeur de l'objet initialisateur. {Post-condition} L'ordonnée du graphique est *ordonnee*, la durée réelle est *duree* et la durée vue est *rythme*.

Les classe CInitialisateurSilenceGraphique et CInitialisateurNoteGraphique héritent de cette classe. CInitialisateurSilenceGraphique ne sera pas expliqué ici vu qu'il ne fait que redéfinir des méthodes virtuelles.

7.4 CInitialisateurNoteGraphique

Données contenues	
CMusiqueNote **note* ; short *nombreLignes* ;	Note représentée par le graphique. Nombre de lignes de portée supplémentaires de la note.
Services offerts à tous	
CInitialisateurNoteGraphique (int *ordonnee*, int *duree*, int *rythme*, short *nombreLignes*)	Constructeur de l'objet initialisateur. {Post-condition} Les données membres correspondant à *ordonnee*, *duree*, *rythme* et *nombreLignes* sont initialisées.

8. Classes de brouillon de graphique horizontal

Les objets dérivés des classes suivantes sont des repères préliminaires à la construction d'objets graphiques horizontaux. En effet, les objets graphiques sont d'abord positionnés les uns après les autres, puis ils sont déplacés horizontalement pour occuper toute la largeur de la portée qui les contient. Leur ordonnée et l'espace qui sépare deux objets graphiques changent. De ce fait, il est nécessaire de postposer la construction des objets graphiques « horizontaux » (c'est-à-dire ceux qui sont fonction des ordonnées de plusieurs autres objets graphiques) après la justification des objets graphiques.

8.1 CBrouillonGraphiqueHorizontal

Données contenues	
short *indiceDebut* ;	Indice, dans l'ensemble des objets graphiques de la partition, du premier des graphiques reliés par ce graphique horizontal.
short *nombreNotes* ;	Nombre de notes incluses dans ce graphique.
int *depassementHaut* ;	Dépassement de ce graphique au-dessus et en dessous de la portée.
int *depassementBas* ;	
CGraphiqueNote **premiereNote* ;	Première et dernière notes englobées dans le graphique horizontal.
CGraphiqueNote **dernieresNote* ;	
Services offerts à tous	
CBrouillonGraphiqueHorizontal (short *debut*, short *nombre*)	Constructeur {Post-condition} Les données membres correspondant à *debut* et *nombre* sont initialisées, les autres sont nulles.
int IndicePremiereNote (CVecteur **vecteurGraphiques*) const ;	{Pré-condition} *vecteurGraphiques* est l'ensemble des objets graphiques de la partition. {Post-condition} On renvoie l'indice dans vecteurGraphiques de la première note englobée par ce graphique horizontal ; on renvoie 0 si on ne la trouve pas.
int IndiceDerniereNote (CVecteur **vecteurGraphiques*) const ;	{Pré-condition} *vecteurGraphiques* est l'ensemble des objets graphiques de la partition {Post-condition} On renvoie l'indice dans *vecteurGraphiques* de la dernière note englobée par ce graphique horizontal, on renvoie la borne supérieure des indices de *vecteurGraphiques* si on ne trouve pas la note.

Les classes CBrouillonGraphiqueLiaison, CBrouillonGraphiqueNOlet et CBrouillonGraphiqueBarre héritent de cette classe.

8.2 CBrouillonGraphiqueLiaison

Données contenues	
CVecteur **graphiques* ;	Ensemble des objets graphiques de la partition
int *ordonneeSommet* ;	Ordonnée du sommet de la liaison par rapport à la première ligne de la portée
float *echelle* ;	Echelle de redimensionnement des objets graphiques

Services offerts à tous	
CBrouillonGraphiqueLiaison (short *debut*, short *nombre*, CVecteur **graphiquesMusicaux*, float *echelle*);	Constructeur de l'objet initialisateur. {Post-condition} Les données membres correspondant à *debut*, *nombreNotes* et *graphiques* sont initialisées.
BOOLEAN MiseAuPropre (const CDecoupagePortee **portee*, int *indiceDebut*, int *indiceFin*)	{Pré-condition} indiceDebut et indiceFin sont les indices de la première et de la dernière note de la liaison dans le vecteur des objets graphiques de la partition. {Post-condition} L'objet graphique relatif à la sous-liaison relative à la portée *portee* et à ce brouillon a été créé et ajouté à l'ensemble des objets graphiques de la partition. Si nécessaire, les dépassements haut et bas du graphique horizontal ont été changés.

8.3 CBrouillonGraphiqueNOlet

Données contenues	
short *n*; float *coefficientRythmique*; int *sommeDistancesSi*;	n du n-olet. Coefficient par lequel on doit diviser la durée vue de la note d'un n-olet pour trouver sa durée effective. Somme des distances entre le corps des notes du n-olet et la ligne du milieu de la portée.
Services offerts à tous	
CBrouillonGraphiqueNOlet (short *n*, float *coefficientRythmique*, short *debut*, short *nombreNotes*)	Constructeur de l'objet initialisateur. {Post-condition} Les données membres correspondant à *debut*, *nombreNotes*, *n*, *coefficientRythmique* sont initialisées.
BOOLEAN AjusteHampe (CVecteur **graphiques*, float *echelle*, long *ordonneeSi*);	{Pré-condition} *graphiques* est l'ensemble des objets graphiques de la partition, *echelle* est l'échelle de redimensionnement des graphiques et *ordonneeSi* est l'ordonnée de la troisième ligne de la portée graphique. {Post-condition} *sommeDistancesSi* est initialisée. Les hampes ont été positionnées en fonction de cette variable : si les notes du groupe sont en majorité plus sous la 3ème ligne de la portée, elles sont orientées vers le haut ; sinon, elles sont orientées vers le bas.

BOOLEAN MiseAuPropre (CDecoupageMesure **mesure*);	{Pré-condition} *mesure* est la mesure de découpage où se trouve le n-olet. {Post-condition} L'objet graphique de n-olet de ce brouillon a été créé et ajouté à l'ensemble des objets graphiques de la partition. Si nécessaire, les dépassements haut et bas du graphique horizontal ont été changés.

8.4 CBrouillonGraphiqueBarre

Données contenues	
short *niveau*;	Niveau dans la hiérarchie des barres (relation d'inclusion).
CVecteur *sousBarres*;	Ensemble des barres de niveau inférieur dans ce groupe.
float *coefficientAngulaire*;	Coefficient angulaire du segment de droite formé par cette barre.
int *nombreGraphiques*;	Nombre de graphiques contenus dans cette barre.
int *sommeDistancesSi*;	Somme des distances entre le corps des notes du n-olet et la ligne du milieu de la portée.
Services offerts à tous	
CBrouillonGraphiqueBarre (short *debut*, short *niveau*);	Constructeur de l'objet initialisateur. {Post-condition} Les données membres correspondant à *début* et *niveau* sont initialisées. Les autres variables membres sont nulles.
BOOLEAN AjouteGraphiques (CDecoupageMesure **mesure*, int *&indiceGraphiqueCourant*, short *&tempsRestant*);	{Pré-condition} *mesure* est la mesure de découpage comprenant le groupe de notes ; *indiceGraphiqueCourant* référence un CGraphiqueNote (la première note de groupe) ; *tempsRestant* est le nombre de ticks restant dans le temps à partir de la note courante. {Post-condition} Toutes les variables membres (sauf *coefficientAngulaire*) de l'objet sont initialisées, les barres de niveau inférieur concernant les notes de cette barre-ci sont créées et initialisées, *indiceGraphiqueCourant* est l'indice de la dernière note de la barre.

void AjusteHampe (const CVecteur **graphiques*, long *ordonneeSi*, float *echelle*);	{Pré-condition} *graphiques* est l'ensemble des objets graphiques de la partition, *echelle* est l'échelle de redimensionnement des graphiques et *ordonneeSi* est l'ordonnée de la troisième ligne de la portée graphique. {Post-condition} *sommeDistancesSi* est initialisée. Les hampes ont été positionnées en fonction de cette variable : si les notes du groupe sont en majorité sous la 3ème ligne de la portée, elles sont orientées vers le haut ; sinon, elles sont orientées vers le bas. La hauteur des hampes est ajustée pour que les extrémités soient alignées ; si nécessaire, les dépassements haut et bas du graphique horizontal ont été changés.
BOOLEAN MiseAuPropre (CDecoupageMesure **mesure*, CBrouillonGraphiqueBarre **barreDessus*);	{Pré-condition} *mesure* est la mesure de découpage où se trouve la barre ; *barreDessus* est la barre de niveau supérieur contenant toutes les notes de celle-ci ; elle n'est pas initialisée pour les barres du premier niveau. {Post-condition} La variable membre *coefficientAngulaire* a été initialisée et l'objet graphique de barre de notes de ce brouillon a été créé et ajouté à l'ensemble des objets graphiques de la partition.

Chapitre 4. Algorithmes

Après avoir décrit les différentes classes d'objets et ce qu'elles permettent de réaliser, nous nous attarderons, dans ce chapitre, à voir comment certaines fonctions sont implémentées.

Nous verrons tout d'abord les algorithmes permettant l'affichage des objets. Ensuite, la plus grande partie de ce chapitre concernera les algorithmes mis en oeuvre afin de dériver les objets graphiques à partir de la valeur du champ musical. Les derniers algorithmes méritant une explication seront développés dans la dernière partie.

1. Affichage

La grande majorité des classes relatives à l'affichage ne font qu'utiliser les services fournis par la classe mère CGraphiqueMusical. Nous allons analyser ci-après l'implémentation de ces services. Ensuite, nous verrons les algorithmes d'affichage d'une liaison, d'un n-olet et d'une note ; les autres, ne présentant que peu de difficultés de compréhension, ne seront pas expliqués, mais se trouvent sur la disquette accompagnant ce travail (fichiers CGra*.*).

1.1 CGraphiqueMusical

La classe CGraphiqueMusical offre une série de services à ses classes filles (cfr chap. 3, section 2.1) : l'affichage d'une image, d'une ligne ou d'un nombre. Quelques problèmes liés au Portability ToolKit (PTK) d'xvt ont été résolus et encapsulés dans cette classe. Le code relatif à celle-ci se trouve en annexe 2.

1.1.1 Affichage d'une image

L'affichage d'une image par le PTK se fait au moyen des ressources suivantes :

- un pixmap (bitmap dans lequel on retrouve l'image) ;
- un rectangle source (partie du pixmap à afficher) ;
- un rectangle cible (endroit, dans la fenêtre, où il faut afficher le pixmap).

Le rectangle cible peut ne pas être de même dimension que le rectangle source. Dans ce cas, l'image est mise aux dimensions du rectangle cible (cfr figure 17). Comme nous

le verrons dans la section 3, seule une translation est nécessaire à ce stade, le redimensionnement ayant été fait au niveau du cache de pixmaps.

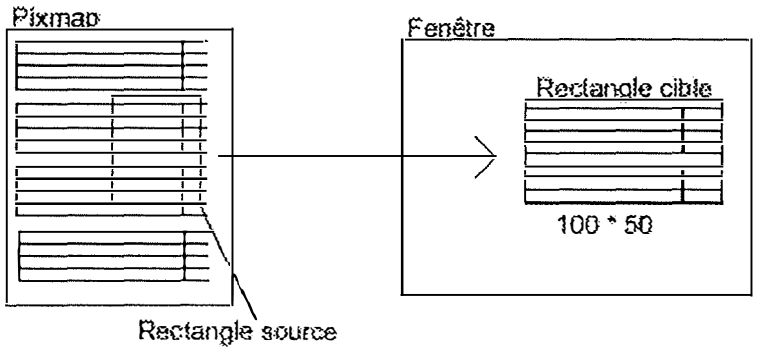


Figure 17: affichage d'une image

Le PTK nous a obligés à afficher les images en deux temps, à l'aide de deux pixmaps différents.

La pixmap vue à l'écran, qui sera appelée par la suite « pixmap couleur », intervient dans la seconde opération. Au préalable, une « pixmap noir-et-blanc » est utilisée. Elle est semblable à la pixmap couleur où tous les pixels non blancs ont été remplacés par du noir.

Après avoir initialisé les deux rectangles grâce aux informations fournies en paramètre, la fonction affiche une première fois la pixmap noir-et-blanc en mode M_CLEAR. Dans ce mode, les pixels noirs de l'image source sont affichés en blanc dans le rectangle cible ; les pixels blancs de l'image source n'influencent pas l'affichage. Ensuite, l'image couleur est affichée en mode M_OR dans le blanc laissé par la première opération. Comme les images couleur et noir-et-blanc sont semblables, à la couleur près, l'image en couleur s'affiche sur fond blanc et semble se superposer aux images déjà affichées (sans mélange (XOR) des couleurs). Ces deux opérations sont assez coûteuses en temps mais aucun moyen plus direct n'est réalisable en utilisant les possibilités du PTK.

Dans la section 3, relative au cache de pixmaps, nous analyserons comment les pixmaps sont créées.

Il est à remarquer que l'image est toujours affichée dans la couleur choisie comme couleur de champ variable par l'utilisateur (champ `fore_color` du record outilsDessin se trouvant dans la partition), sauf les cercles barrés d'avertissement (plus d'informations dans la section 2.2.1) qui sont toujours rouges.

1.1.2 Affichage d'une ligne

L'affichage d'une ligne procède par les deux mêmes étapes que l'affichage d'une image (effacement, puis dessin). Le PTK, lorsque le trait est plein (non pointillé), arrondit les

extrémités de la ligne et le résultat est différent d'une plate-forme à l'autre. Pour pallier ce problème, la méthode AffichePolygone est utilisée. Elle transforme un trait en un polygone et l'affiche par la procédure xvt_dwin_draw_polygone.

1.1.3 Affichage d'un nombre

Le PTK offre la possibilité d'afficher du texte sur fond transparent. Le problème de l'effacement ne se pose donc pas ici. Cette méthode ne présente pas de difficulté de compréhension.

1.2 CGraphiqueLiaison

Le code relatif à cette classe se trouve en annexe 3.

Rappelons qu'une liaison est une courbe reliant deux notes d'une même portée. Pour afficher une courbe par l'intermédiaire du PTK, il est nécessaire de spécifier un rectangle et deux points. La courbe affichée sera une partie d'ellipse. Cette ellipse a un axe horizontal et est tangente aux côtés du rectangle en leur milieu. L'arc est défini par les deux points qui sont son origine et son extrémité ; le sens de traçage est le sens contraire à celui des aiguilles d'une montre.

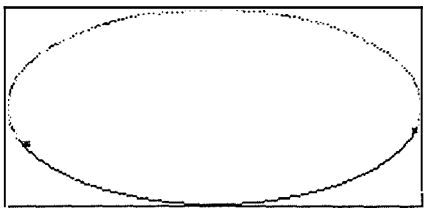


Figure 18: rectangle, ellipse, points = arc

Après avoir initialisé toutes les variables nécessaires à l'affichage, on calcule les coordonnées des points et du rectangle en fonction de l'écran. Le calcul est

```
debutX = positionFenetre - ovale.left + pointDebut.h.
```

Il mérite quelques explications car ce calcul est différent de celui utilisé pour l'affichage des autres graphiques. Le fait est que les coordonnées des points pointDebut et pointFin sont exprimés ici en coordonnées relatives à la fenêtre et non en coordonnées relatives à l'objet, comme c'est le cas des autres graphiques. debutX doit donc être égal à pointDebut.h + décalageGauche. On peut le vérifier :

```
debutX = positionFenetre - ovale.left + pointDebut.h
debutX = positionX + decalageGauche - ovale.left
         + pointDebut.h
```

Or, il est facile de constater (dans l'annexe 6, à la fin de la méthode MiseAuPropre de CBrouillonGraphiqueLiaison) que l'abscisse de l'objet (positionX) est égale à la coordonnée gauche du rectangle (ovale.left). Et donc

```
debutX = decalageGauche + pointDebut.h
```

Lorsque le calcul des coordonnées est terminé, il reste à afficher l'arc par un xvt_dwin_draw_arc.

1.3 CGraphiqueNOlet

En regardant la méthode d'affichage des n-olets, on remarque d'emblée des options de précompilation. Leur résultat est d'afficher les n-olets en pointillés sur les plates-formes Unix et Windows et en trait continu sur MacIntosh. C'est le résultat du test de compatibilité effectué sur MacIntosh. En effet, il était prévu de toujours afficher les lignes du n-olet en pointillés, mais lors de ce test, il s'est avéré que la première barre du n-olet est systématiquement affichée en trait plein sur MacIntosh. Après une série de tests, il est supposé qu'il s'agit d'une erreur dans le PTK. Il faut donc suivre son évolution et attendre la sortie d'une nouvelle version pour éventuellement pallier ce problème. A ce jour, aucun test sur plate-forme Unix n'a été réalisé.

La figure suivante illustre les constantes choisies et la position des différents points remarquables.

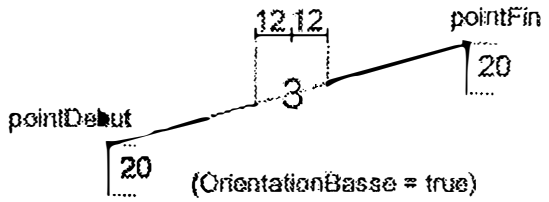


Figure 19: n-olet

1.4 CGraphiqueNote

L'affichage des graphiques de note n'est pas difficile à comprendre. Les commentaires accompagnant le code (cfr annexe 4) pourraient presque suffire. Ci-dessous se trouve un tableau reprenant les différentes étapes d'affichage d'une note : il permet de retrouver facilement la méthode correspondante et la classe dans laquelle elle est implémentée.

Étape	Méthode invoquée	Classe mère où est implémentée la méthode
Dessin du corps de la note	AfficheImage	CGraphiqueMusical
Traçage des lignes de portée supplémentaires	AfficheLignes	CGraphiqueNote
Traçage de la hampe	AfficheHampe	CGraphiqueNoteHampe
Dessin des moustaches	AfficheMoustaches	CGraphiqueNoteMoustachue
Dessin des points	AffichePoints	CGraphiquePointe

2. Découpage

Dans cette section, le fonctionnement des classes qui transforment le code du champ musical en objets graphiques sera analysé. Nous verrons, dans un premier temps, les différentes étapes de cette transformation ; ensuite, les étapes seront expliquées une à une. Les indications en marge du texte visent à attirer l'attention sur l'explication de méthodes, de variables, dépassant souvent le cadre de la section où elles se trouvent, et qu'il n'est pas possible de repérer via la table des matières. Les annexes 5 et 6 contiennent les listings des fichiers concernés par le découpage.

2.1 Étapes

Les différentes étapes du découpage se retrouvent dans la méthode Ajoute de CDecoupagePartition. Cette sous-section vise à illustrer le but de ces étapes. Leur fonctionnement sera développé par la suite.

Première étape : création et remplissage des portées (RemplisPortee)[19]

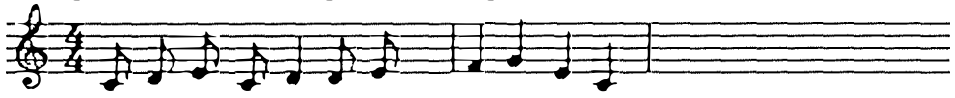


Figure 20: fin de la première étape de découpage

Lors de cette étape, les codes du champ musical sont analysés et placés, sous forme de graphiques musicaux, dans des portées. L'espace entre graphiques est constant, les barres groupant les notes, les liaisons et les n-olets n'apparaissent pas encore.

[19] les figures représentées dans cette section sont des représentations de ce qui se passe en mémoire « comme si on effectuait un affichage à ce moment ». L'affichage se faisant toujours après le découpage, seule la figure correspondant à la dernière étape est effectivement affichée.

Deuxième étape : justification des portées (Justifie)

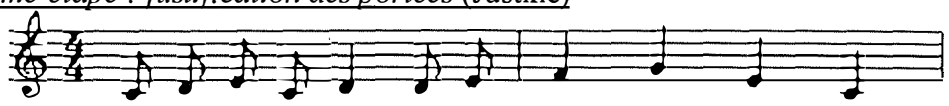


Figure 21: fin de l'étape de justification des portées

Les notes sont espacées proportionnellement à leur durée et les graphiques de chaque portée complète sont justifiés pour occuper tout l'espace disponible.

Troisième étape : regroupement des notes par temps, ajout des graphiques de n-olets et de liaisons (AjouteBarreNotes, AjouteNOlets, AjouteLiaisons)

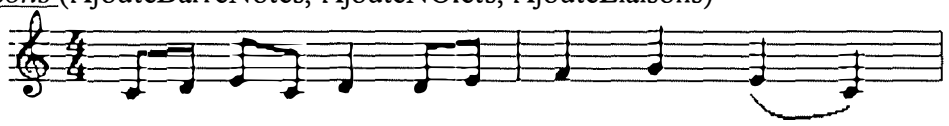


Figure 22: fin de l'étape d'ajout des graphiques horizontaux

Ces graphiques ne sont pas créés avant la deuxième étape car leur position dépend de la position horizontale de plusieurs notes de la mélodie. Afin d'éviter le recalcul fastidieux des coordonnées de ces objets et de leur inclinaison (pente), celles-ci ne sont calculées que lorsque la coordonnée horizontale de l'objet est définitive. En attendant, des « brouillons » de ces objets sont stockés (cfr infra, le traitement des codes associés à ces objets).

Le traitement des barres reliant des groupes de notes et des n-olets est plus aisé que celui des liaisons. En effet, les deux premiers (groupe de notes et n-olet) ne se rencontrent qu'à l'intérieur d'une mesure, tandis qu'une liaison peut déborder les mesures et même concerner plusieurs portées.

Quatrième étape : écartement des portées (EcartePortee)

Avant cette étape, les objets graphiques sont positionnés par rapport au point supérieur gauche de leur portée. Si la mélodie s'affichait à ce moment, toutes ses portées se superposeraient en haut de la fenêtre ! Cette étape applique une translation aux graphiques afin que l'espace entre portées soit minimal et que les graphiques de deux portées ne se chevauchent pas.

Cinquième étape : création du fond de mélodie

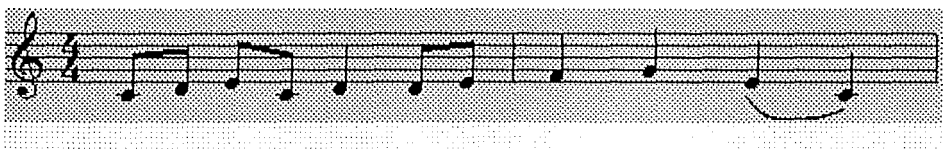


Figure 23: résultat final

La dernière étape consiste en l'ajout d'un fond de couleur (choisie par l'utilisateur) qui englobe tous les graphiques de cette mélodie.

2.2 Création et remplissage des portées

2.2.1 Remplissage au niveau de la partition de découpage

Comme on l'a vu au chapitre 2, chaque mélodie commence par un code d'en-tête. Celui-ci indique la version d'encodage de la mélodie et le tempo. Il est analysé par la méthode RemplisPortee de la classe CDecoupagePartition.

Afin d'indiquer la version d'encodage à l'utilisateur, des signaux d'avertissements sont ajoutés en haut de la mélodie. Il sont au nombre de trois :

Le premier (une croche barrée) signale que les rythmes affichés ne sont pas pertinents, le deuxième (un bémol barré) que l'armature n'est pas pertinente et le dernier (des mesures barrées) que le placement des barres de mesure a été réalisé manuellement (n'est pas géré automatiquement sur base de l'indication de mesure).

Après le placement de ces signaux, l'indication de tempo est ajoutée.

Finalement, les codes suivants sont analysés dans une boucle qui crée une CDecoupagePortee et la remplit.

2.2.2 Remplissage au niveau de la portée de découpage

Le premier graphique créé par la portée de découpage est sa portée graphique. Ensuite, elle crée et remplit ses différentes mesures de découpage. La première mesure doit comporter une clé et une armature ; directement après sa création, on lui demande donc d'ajouter l'armature à la liste de ses objets graphiques.

A la sortie de la boucle créant et remplissant les mesures, 5 situations peuvent se présenter.

- Situation 1 : si le dernier code de la mélodie a été traité et que la mesure est complète, le découpage est terminé et la valeur TRUE est renvoyée à la fonction appelante ;
- Situation 2 : si le dernier code de la mélodie a été traité mais que la mesure n'est pas complète, il faut inclure cette mesure dans la portée même si elle est incomplète et renvoyer la valeur TRUE (car si la mesure est incomplète, la boucle a été interrompue via le BREAK et la mesure n'est pas incluse dans le vecteur des mesures) ;
- Situation 3 : si le dernier code n'a pas été traité mais que la portée courante contient déjà des mesures, cette mesure n'est pas incluse dans la portée, elle sera prise en charge par la portée suivante ;
- Situation 4 : si le dernier code de la mélodie n'a pas été traité, que la mesure non complète est la première de la portée courante et qu'aucun code n'a été traité dans cette portée, nous sommes dans une situation de découpage impossible (p.ex., parce que la largeur disponible est trop petite pour traiter un seul code) ; il ne reste plus qu'à renvoyer FALSE pour l'indiquer à la fonction appelante ;

- Situation 5 : si le dernier code de la mélodie n'a pas été traité, que la mesure incomplète était la première mais qu'au moins un code a été traité, un graphique de continuateur de mesure (deux lignes obliques parallèles, indiquant que la mesure se poursuit sur la portée suivante) est ajouté à cette mesure qui est finalement ajoutée au vecteur de mesures.

2.2.3 Remplissage au niveau de la mesure de découpage

Ajoute La méthode Ajoute de CDecoupageMesure boucle aussi longtemps qu'il existe des codes à analyser, qu'il reste de la place dans la mesure et que la mesure n'est pas complète. A l'intérieur de la boucle, le code est traité et des graphiques sont ajoutés. Les sous-sections suivantes reprennent chaque type de code et expliquent comment ils sont traités (les traitements concernant les codes de changement de clé, de changement de mesure, de barre de mesure ne seront pas expliqués : les commentaires annexés au code suffisent à leur compréhension). La dernière sous-section explique la méthode AjouteSignetsBarresNotes qui clôture la méthode d'ajout de graphiques dans la mesure de découpage.

2.2.3.1 Code de silence

2.2.3.1.1 Traitement du code

La durée[20] du silence est extraite du paquet. La durée effectivement affichée est calculée en fonction du fait que le silence fait partie ou non d'un n-olet. Nous reviendrons à ce problème lorsque nous analyserons le traitement d'un code de n-olet.

2.2.3.1.2 Ajout du graphique

Pour l'ajout du graphique, la classe CInitialisateurGraphiqueSilence est utilisée. Elle ne figure pas en annexe car son code ne présente aucune difficulté, mais bien sur la disquette d'accompagnement. Pour plus d'explications concernant cette classe, veuillez vous reporter au chapitre précédent, section 7.

AjouteGraphique La méthode privée AjouteGraphique vérifie qu'il reste assez de place dans la mesure pour insérer un graphique, insère ce graphique à l'endroit demandé (en tenant compte des espaces demandés avant et après celui-ci) et met à jour la largeur de la mesure.

2.2.3.2 Code de changement de ton

2.2.3.2.1 Traitement du code de changement de ton

Dans cette méthode, les altérations courantes sont modifiées en fonction du nouveau ton ; l'ancien ton est renvoyé en résultat.

---

[20] les durées sont toujours exprimées en ticks.

alterationsCourantes est un tableau de sept entiers qui, dans la pratique, ne peuvent prendre que les valeurs -1, 0, 1. Les sept indices du tableau sont les notes (do, ré, mi, fa, sol, la et si). La valeur d'un élément du tableau indique l'altération de la note. Ce tableau est utilisé non seulement pour connaître et changer le ton de la mélodie, mais également pour déterminer si une note doit être précédée d'une altération accidentelle (visible) ou si son altération fait partie du ton courant (pas visible). Les différentes sortes d'altérations ont été expliquées au chapitre premier, section 1.1.

alterations-Courantes

2.2.3.2.2 Ajout des graphiques

L'ajout des graphiques de changement de ton doit se faire dans les règles précisées au chapitre 1, section 1.1, figure 5. Pour cela, la méthode AjouteChangementTon appelle les méthodes AjouteTonalité et/ou AjouteBecarreApres.

Les commentaires liés à ces méthodes suffisent pour en comprendre le fonctionnement. Nous allons néanmoins nous attarder sur une méthode de la classe CMusiqueCle[21] qui sert à calculer l'ordonnée des altérations : HauteurAlterations. En guise de rappel du chapitre 1, notons que les clés influencent la hauteur des notes et que les altérations d'armature se rapportent à une note particulière prise dans la suite [fa, do, sol, ré, la, mi, si] pour les dièses et [si, mi, la, ré, sol, do, fa] pour les bémols. De ces deux faits, il découle que la position dans la portée des altérations d'armature est différente selon la clé utilisée. Ce problème ne peut pas se résoudre au moyen d'une simple translation car dans une certaine clé, certaines altérations d'armature peuvent être situées une octave plus haut que dans une autre clé ! Nous avons résolu ce problème en gardant, pour chaque clé, un tableau indiquant, pour chaque altération, son ordonnée supérieure par rapport à la ligne supérieure de la portée. C'est ce tableau qui est consulté avant la création des graphiques d'altérations d'armature.

Hauteur-Alterations

2.2.3.3 Code de n-olet

Le traitement associé à ce code est assez simple. On crée un brouillon de n-olet, en l'initialisant correctement. Ensuite, il est ajouté à l'ensemble des brouillons de n-olets de la mesure.

La variable nombreNotesNOlet de la classe CDecoupageMesure est plus intéressante. Celle-ci renseigne du fait que l'on se trouve dans un n-olet ou pas. Elle permet de calculer la durée vue d'une note ou d'un silence alors que le code ne nous donne que la durée effective (entendue). Pour calculer la durée vue d'une note d'un n-olet, il faut multiplier la durée effective par le coefficient rythmique du brouillon correspondant au dernier n-olet rencontré.

nombre-Notes-Nolet

[21] le listing des classes CMusique se trouve en annexe 7.

2.2.3.4 Code de liaison

Ce code est également stocké, en vue d'un traitement ultérieur, dans une liste de brouillons qui se trouve dans la partition de découpage (cfr 2.4.3).

2.2.3.5 Code de note

2.2.3.5.1 Traitement du code de note

Le traitement a pour but de transformer le code de note en un objet de type CMusiqueNote qui lui correspond. Dans un premier temps, la hauteur de la note est déterminée mais est toujours exprimée soit comme une note naturelle, soit comme une note diésée. Ces deux sous-ensembles des notes (do, do#, ré, ré#, mi, fa, fa#, sol, sol#, la, la# et si) couvrent les douze hauteurs comprises dans une octave. Dans un second temps, l'altération correcte de la note est déterminée grâce aux deux bits d'altération contenus dans le code de note. Le tableau suivant offre quelques exemples de cette transformation :

1er temps	altération souhaitée	2nd temps
Do#	bémol	ré***b***
Do#	dièse	Do#
Do	bémol	Ré***bb***
Do	dièse	Si#

2.2.3.5.2 Ajout des graphiques

Avant d'ajouter les graphiques (altération et note), l'ordonnée de la note est calculée. Pour ce faire, la hauteur vue de la note est calculée. Elle est exprimée (par convention) par rapport à la clé de sol 2ème ligne.

Ensuite, si nécessaire, une altération accidentelle est ajoutée. Dans ce cas, nous pouvons remarquer que les altérations courantes de la mesure sont modifiées. En effet, comme nous l'avons vu au chapitre 1, une altération accidentelle influence toutes les notes de sa hauteur pendant toute la durée de la mesure.

Finalement, le graphique de la note est ajouté. Comme pour les silences, cela se fait par l'intermédiaire d'un CInitialisateurGraphique afin d'encapsuler un traitement fastidieux (la recherche de la durée vue de la note à partir de sa durée en ticks).

2.2.3.6 Méthode AjouteSignetsBarresNotes

Cette méthode va analyser le contenu de la mesure et créer les brouillons de barres reliant les notes. Ceux-ci seront transformés en graphiques de barre dès que les coordonnées exactes des graphiques seront connues.

Elle calcule tout d'abord la durée d'un temps en fonction de la mesure courante (cette valeur est exprimée en 128èmes de noire et ne varie donc pas avec le tempo). Elle analyse tous les graphiques musicaux, ne retenant que les notes et les silences. Chaque fois qu'elle rencontre une note ou un silence, le nombre de 128èmes de noire restant dans le

temps courant diminue d'un nombre de 128èmes de noire équivalent à la durée de la note ou du silence rencontré. Si le temps courant devient négatif, on lui ajoute le nombre de 128èmes de noire contenus dans un temps (car une mesure peut contenir plusieurs temps).

Lorsque la note rencontrée possède une ou plusieurs moustaches, la méthode TraiteBarreNote est appelée. Celle-ci va soit créer les brouillons de barres correspondant à toutes les moustaches des notes reliées à la note courante, soit, si la note est orpheline (n'est reliée à aucune autre), transformer les moustaches logiques de la note en moustaches graphiques. Cette méthode fait appel à la méthode AjouteGraphiques de CBrouillonGraphiqueBarre qui va être analysée dans les paragraphes suivants. **TraiteBarreNote**

Précisons d'emblée la notion de niveau de barre. Lorsqu'une note possède plusieurs moustaches (p.ex. une double croche) et qu'elle est groupée avec d'autres, le groupement se fait avec autant de lignes parallèles qu'il y a de moustaches à la note (cfr chapitre 1, figure 9). Le niveau d'une barre est l'ordre de la moustache qu'elle remplace (niveau 1 : moustache supérieure, ...).

On fait appel à AjouteGraphiques lorsqu'on se trouve sur la première note d'un groupe relié par une barre. La méthode va traiter toutes les notes groupées par cette barre et, si c'est nécessaire, créer des barres de niveaux inférieurs. La boucle principale se déroule tant que la barre n'est pas totalement traitée (on n'est pas arrivé à la fin du temps et le nombre de moustaches de la note courante est plus grand que le niveau de barre). Trois situations peuvent alors être rencontrées : **AjouteGraphiques**

1. La durée de la note courante est plus grande que le temps restant, la barre courante est donc totalement traitée ;

2. La situation 1 n'est pas observée et le nombre de moustaches est égal au niveau de la barre, le temps restant est alors diminué et cette note est ajoutée à la barre ;

3. La situation 1 n'est pas observée et le nombre de moustaches est supérieur au niveau de la barre, une barre de niveau juste inférieur est alors créée et remplie (en appelant sa méthode AjouteGraphiques), et les notes de cette barre sont ajoutées à la barre courante.

On avance alors sur la note suivant la note ou le groupe de notes traité (en faisant attention de diminuer le temps restant lorsqu'on rencontre un silence).

Notons pour terminer que seules les barres de niveau supérieur sont stockées dans le vecteur de la mesure de découpage. Les références aux barres de niveaux inférieurs se retrouvent dans un vecteur de la barre qui les a créées.

2.3 Justification des portées

2.3.1 Au niveau de la portée de découpage

La méthode Justifie de CDecoupagePortee va affecter la largeur encore disponible de la portée aux mesures qu'elle contient. Elle utilise pour cela la méthode ArrangeGraphi- **Justifie**

ques(decalage, supplement) de CDecoupageMesure, où *decalage* est la somme des suppléments ajoutés aux mesures précédentes et où *supplement* est le nombre de pixels à ajouter à la mesure courante ; ce *supplement* est calculé comme la largeur encore disponible répartie dans les mesures restantes, pour éviter les erreurs d'arrondi.

2.3.2 Au niveau de la mesure de découpage

Arrange-Graphiques

Dans un premier temps, un nombre de « pixels idéaux » est calculé en fonction du décalage idéal (cfr chap.3, section 2.1) de chaque graphique contenu dans la mesure. Comme il a été expliqué au chapitre 1, les graphiques peuvent être plus ou moins écartés les uns des autres (en fonction de leur durée, par exemple). Le décalage idéal d'un graphique est le nombre de pixels que l'on peut ajouter à sa droite. Il n'a pas de signification absolue mais est pertinent lorsqu'on compare deux décalages de graphiques différents. La situation la plus esthétique (idéale) voudrait que tous les graphiques de la mesure soient décalés d'un nombre de pixels égal à leur DécalageIdeal multiplié par une constante commune. Elle ne se présente que très rarement car le nombre de pixels que l'on doit ajouter est imposé par la portée de découpage (cfr supra). Ces pixels vont donc être répartis au mieux.

Pour chaque graphique, le décalage à faire subir à ceux qui sont à sa droite est calculé par la formule suivante :

$$décalage_i = \frac{\sum_{j=0}^{i} décalageIdeal_j * pixelsSupplémentaires}{\sum_{j=0}^{n} décalageIdeal_j} - \sum_{j=0}^{i-1} décalage_j$$

Cette formule nous assure que tous les pixels supplémentaires seront ajoutés car le calcul du dernier décalage donne

$$décalage_n = pixelsSupplémentaires - \sum_{j=0}^{n-1} décalage_j$$

$$\Rightarrow 0 = pixelsSupplementaires - \sum_{j=0}^{n} décalage_j$$

De plus, $\frac{\sum_{j=0}^{i} décalageIdéal_j}{\sum_{j=0}^{n} décalageIdéal_j}$ nous assure que les graphiques seront espacés proportionnellement à leur décalage idéal.

2.4 Ajout des graphiques horizontaux

2.4.1 Barres groupant des notes

Rappelons que nous avons déjà créé les brouillons des graphiques de barres (cfr 2.2.3.6) à la fin du traitement de chaque mesure. Il ne reste plus à effectuer que les opérations dépendant de l'ordonnée des notes (qui, depuis la justification des portées, est définitive).

L'ajout des barres de notes se fait au niveau de la mesure de découpage par l'invocation des méthodes AjusteHampes et MiseAuPropre des CBrouillonGraphiqueBarreNote. Le résultat de AjusteHampes est que 1) toutes les hampes des notes du groupe sont orientées dans la même direction (bas ou haut) et que 2) les extrémités de ces hampes sont alignées. MiseAuPropre transforme les brouillons en graphiques de barre et les ajoute au vecteur de graphiques de la partition.

2.4.1.1 AjusteHampes

Après avoir repéré les première et dernière notes de la barre et calculé le coefficient angulaire de la droite qui les relie, on calcule 1) la somme des distances des notes de la barre à la troisième ligne de la portée (afin de déterminer l'orientation des hampes du groupe) et 2) les distances minimum et maximum de ces notes à la droite reliant les notes extrêmes du groupe (nous en verrons l'utilité plus loin).

L'orientation (haute ou basse) des hampes des notes est déterminée par le signe de la somme des distances de ces notes à la troisième ligne de la portée. Si cette somme est négative (notes plus hautes que la troisième ligne), les hampes sont orientées vers le bas. Sinon, elles sont orientées vers le haut.

La hauteur des hampes est calculée pour qu'elles soient toutes plus grandes ou égales à la constante HAUTEUR_HAMPE_DEFAUT. Regardons les cas des hampes hautes (la hauteur des hampes basses se calcule de manière similaire).

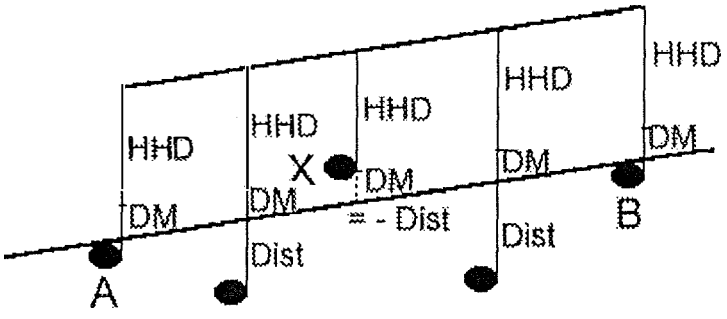


Figure 24: hauteurs des hampes des notes d'un groupe

Appelons AB la droite reliant la première et la dernière note du groupe et appelons X la note la plus haute par rapport à cette droite. La figure ci-dessus illustre les variables utilisées ci-après ; voici la signification des abréviations : HHD pour HAU-

TEUR_HAMPE_DEFAUT, DM pour distanceMaximum, Dist pour distance. La hauteur des hampes hautes est calculée par la formule

$$hauteurHampe = distanceMaximum + HAUTEUR\_HAMPE\_DEFAUT + distance$$

- où *distanceMaximum* est la distance séparant la droite AB et la note X qui a été calculée auparavant ; et
- où *distance* est la distance séparant la note (associée à la hampe courante) de la droite AB.

Cette formule donne à la note X une hampe de hauteur égale à HAUTEUR_HAMPE_DEFAUT (car distanceMaximum = -distance) et donne aux autres notes une hauteur de hampe plus grande afin que les extrémités soient alignées et que la droite les reliant ait un coefficient angulaire égal au coefficient angulaire de AB.

2.4.1.2 MiseAuPropre

Deux cas peuvent être rencontrés : la barre n'est reliée qu'à une seule note (comme les barres de niveau 2 de la figure 25) ou elle relie plusieurs notes (comme la barre de niveau 1 de la figure 25)

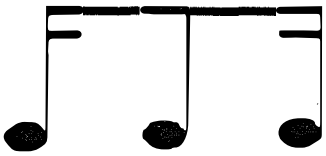


Figure 25: groupe de notes

Dans le premier cas, la barre a une largeur de 5 pixels et est tangente à la hampe, à son extrémité gauche si la barre de niveau supérieur commence à la note concernée ; sinon, elle est tangente à son extrémité droite.

Comme l'illustre la figure 26, si la hampe est haute, il est nécessaire de translater la barre horizontalement.

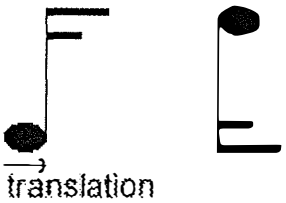


Figure 26: translation des barres pour les hampes hautes

Le cas des barres reliant plusieurs notes n'est pas expliqué dans ce travail, les commentaires annexés au code permettent une bonne compréhension de l'algorithme. Si-

gnalons qu'à la fin de ce traitement, les barres de niveau inférieur concernant des notes incluses dans la barre courante sont mises au propre.

2.4.2 N-Olets

Un brouillon de n-olet a été créé chaque fois qu'un code de n-olet était rencontré (cfr section 2.2.3.3). Il suffit donc d'ajuster l'orientation des hampes et de créer le graphique du n-olet.

La méthode AjusteHampes est semblable à celle qui concerne les barres de notes, quoique beaucoup plus simple, vu qu'on ne modifie pas la taille des hampes,. On se reportera utilement à la section 2.4.1.1.

La création du graphique se fait en deux étapes. La première consiste à calculer les coordonnées du graphique en fonction de la première et de la dernière note (en ignorant la position des autres). La figure ci-dessous illustre les quatre cas possibles d'orientation et de pente du graphique de n-olet. Avec les commentaires annexés au code, elle devrait permettre la compréhension de cette partie.

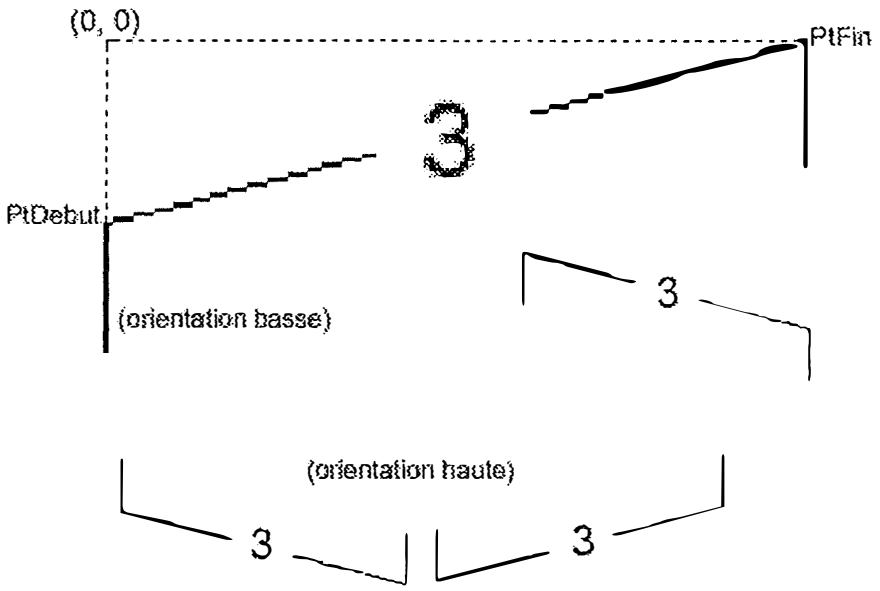


Figure 27: orientation, coordonnées, pointDebut et pointFin d'un n-olet

La seconde partie de la méthode consiste à vérifier qu'aucune note englobée par le n-olet ne vient se dessiner sur le graphique du n-olet. Pour ce faire, on vérifie que l'écart entre l'ordonnée maximale de la note et le point de même abscisse du graphique du n-olet est d'au moins 20 pixels (mis à l'échelle). Si ce n'est pas le cas, le graphique de n-olet est translaté afin d'obtenir cet écart. Remarque : le code est malgré tout un peu plus complexe car il faut tenir compte du cas où le graphique du n-olet se trouve au-dessus des notes et celui où il est en dessous.

2.4.3 Liaisons

Les brouillons de liaison sont stockés au niveau de la partition de découpage. En effet, une liaison peut déborder d'une mesure, même se trouver représentée sur plusieurs portées (cfr la notion de sous-liaison au chapitre 1). Nous verrons, par la suite, comment la partition répartit les liaisons dans les portées, comment une liaison est traitée au niveau d'une portée et comment le brouillon de liaison crée le graphique de la liaison à laquelle il est associé.

2.4.3.1 Au niveau de la partition

La partition va demander à ses premières portées de traiter chaque brouillon de liaison (c'est-à-dire qu'elle itère sur ses portées en suivant leur ordre). Normalement, on doit toujours avoir terminé le traitement d'une liaison lorsqu'on l'a soumise à la dernière portée de la partition.

2.4.3.2 Au niveau de la portée

Par rapport à une portée, une liaison peut avoir cinq positions :

Cas simples :

1) La note de début et la note de fin de la liaison se trouvent dans cette portée ;

2) La note de début et la note de fin se trouvent dans des portées suivant la portée courante.

Cas plus complexes :

1) La note de début de la liaison se trouve dans la portée mais la note de fin est dans une portée suivante ;

2) La note de fin de la liaison se trouve dans la portée mais la note de début était dans une portée précédente ;

3) La note de début de la liaison se trouve dans une portée précédente et la note de fin se trouve dans une portée suivante.

Ces éventualités correspondront à des appels à la méthode MiseAuPropre (la méthode chargée de la création du graphique de liaison, encapsulée dans la classe CBrouillonGraphiqueLiaison) avec les paramètres suivants :

TraiteLiaison

Cas 1	MiseAuPropre (indiceDebut, indiceFin)
Cas 2	pas d'appel
Cas 3	MiseAuPropre (indiceDebut, -1)
Cas 4	MiseAuPropre (-1, indiceFin)
Cas 5	MiseAuPropre (-1, -1)

Cette sélection se fait dans la méthode TraiteLiaison.

2.4.3.3 Création du graphique de liaison

La création du graphique de la liaison se fait en quatre étapes : on détermine tout d'abord l'orientation de la liaison (passe-t-elle au-dessus ou en dessous des notes ?) ; ensuite, les coordonnées des points de début et de fin de la liaison sont calculées ; avant de créer effectivement le graphique, il reste à calculer le carré englobant la liaison.

Comme nous l'avons remarqué à la section 1.2 (graphique de liaison), la liaison est une partie d'une ellipse tangente aux quatre côtés d'un rectangle. Pour des questions de facilité et d'efficacité, nous avons essayé le cas où l'ellipse est un cercle (le rectangle est un carré). Le résultat de ce test étant concluant, nous avons adopté cette solution. En pratique, donc, une liaison est un arc de cercle dont le cercle est tangent aux quatre côtés d'un carré.

2.4.3.3.1 Mise en évidence de l'orientation de la liaison

DetermineOrientation

Cette mise en évidence est réalisée par la méthode DetermineOrientation de CBrouillonGraphiqueLiaison. L'orientation est déterminée par les règles suivantes :

1) si la première note de la liaison possède une hampe, l'orientation de la liaison est l'opposé de celle de cette hampe[22] ;

2) sinon, si la dernière note de la liaison possède une hampe, l'orientation de la liaison est l'opposé de celle de cette hampe ;

3) sinon, l'écart maximum entre le point le plus haut de chaque note et la troisième ligne de la portée, et l'écart maximum entre le point le plus bas de chaque note et la troisième ligne de la portée sont calculés. Si le premier est plus grand que le second, la liaison sera positionnée sous les notes, sinon, elle le sera au-dessus des notes.

Cette règle n'est certainement pas parfaite. La théorie musicale n'en impose en fait aucune : c'est celui qui la dessine qui juge l'aspect esthétique de sa liaison. Ce jugement se base sur des paramètres et des règles complexes dont l'« élicitation » sort du cadre de ce travail. Il est à espérer que la règle fournie à l'ordinateur minimisera le nombre de situations où la liaison est placée de manière tout à fait inesthétique. Les tests réalisés à ce jour paraissent positifs.

2.4.3.3.2 Calcul des coordonnées

L'ordonnée des points de début et de fin de la liaison est trois pixels au-dessus de l'ordonnée supérieure respectivement de la première et de la dernière note (si la liaison se situe au-dessus des notes) ou trois pixels en dessous de l'ordonnée inférieure de la première et de la dernière note (si la liaison se situe en dessous des notes). L'abscisse est l'abscisse du milieu de la note sauf dans les cas où la liaison ne s'arrête pas sur une note

[22] c'est-à-dire qu'elle se trouve au-dessus des notes si l'orientation de la hampe est basse, et inversement.

de la portée (cfr supra, cas complexes) ; elle est alors de 50 pixels (mis à l'échelle) à gauche ou à droite de la première ou dernière note selon qu'il s'agit du point de début ou de fin de la liaison.

2.4.3.3.3 Calcul des paramètres du carré de la liaison

Calcule-Carre Les figures suivantes illustrent les calculs réalisés dans la méthode CalculeCarre. A et B sont les points de début et de fin de la liaison. J, K, L, M et N sont des points associés aux notes englobées par la liaison. Ils sont un peu au-dessus du point supérieur de la note si la liaison passe au-dessus des notes, un peu en dessous du point inférieur sinon. Nous supposerons, ci-après, que la liaison doit passer sous les notes. Le traitement du cas inverse est très semblable.

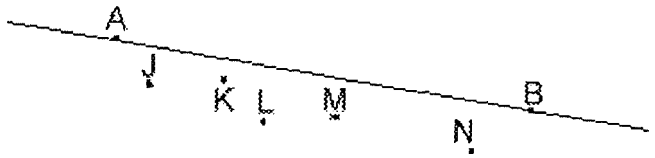


Figure 28: liaisons : droite AB et points correspondant à chaque note

Après avoir calculé le coefficient angulaire de la droite AB, on recherche :

1. $Y \in \{J, K, L, M, N\}$ tel que (coefficient angulaire AY) =

   $\max_{X \in \{J,K,L,M,N\}}$ (coefficient angulaire AX). Dans la situation de la figure 28, Y = J.

2. $Z \in \{J, K, L, M, N\}$ tel que (coefficient angulaire BZ) =

   $\min_{X \in \{J,K,L,M,N\}}$ (coefficient angulaire BX). Dans la situation de la figure 28, Z = N.

Il faut remarquer que les coefficients angulaires sont de signe opposé à ceux côtoyés en géométrie euclidienne. En effet, le point de coordonnée (0, 0) se situe en haut, à gauche et non en bas à gauche.

Toutes les notes se trouvent dans le cadran supérieur de la division du plan formée par les droites AY et BZ. Un arc passant par A, B et par l'intersection de AY et BZ passera donc sous toutes les notes.

Afin d'éviter de calculer des liaisons dont la courbure serait inadaptée à l'orientation, comme ce serait le cas dans la situation représentée à la figure 29, on ajoute une valeur qui intervient dans la recherche des maximum et minimum. Il s'agit d'un point (P) qui se situe à quelque cinq pixels sous le milieu du segment [AB]. Ce point est l'intersection de deux droites. La première passe par A et sa distance à la droite AB au point B est de 10

pixels. La seconde passe par B et sa distance à la droite AB au point A est de 10 pixels. La tangente de l'angle formé par ces droites et AB est égale au rapport entre la longueur du côté adjacent de l'angle et celle de son côté opposé (angles et rapports remarquables dans un triangle rectangle).

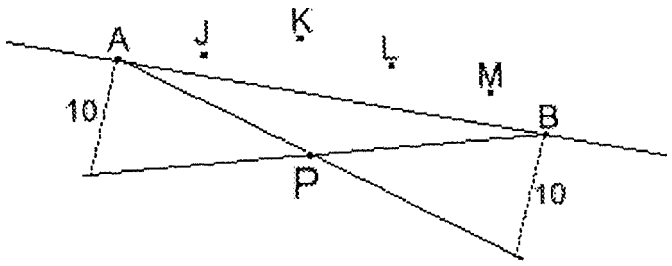


Figure 29: point P

Il faut encore calculer le coefficient angulaire des droites AP et BP. Nous connaissons le coefficient angulaire de AB (qui est égal à la tangente de l'angle formé par la droite et l'horizontale) et la tangente de l'angle $A\hat{B}P$. Le coefficient de AP est donc égal à $\tan(\arctan x + \arctan y)$ que l'on peut développer en $\frac{\tan\arctan x + \tan\arctan y}{1 - (\tan\arctan x \tan\arctan y)}$ qui, en simplifiant, donne $\frac{x+y}{1-xy}$. Le calcul du coefficient angulaire de BP est identique aux signes près.

Connaissant Y et Z, on calcule l'intersection de AY et BZ, soit X (cfr figure 30). La formule utilisée pour trouver cette intersection a été dérivée des équations des deux droites selon le calcul suivant :

$$\begin{cases} y = CA_A(x - x_a) + y_a \\ y = CA_B(x - x_b) + y_b \end{cases}$$ équations des droites AY et BZ

CA_A = coefficient angulaire de AY

CA_B = coefficient angulaire de BZ

(x_a, y_a) = coordonnées du point A

(x_b, y_b) = coordonnées du point B

Les deux membres de droite de ce système étant égaux, ce système peut se transformer en l'équation suivante :

$$CA_A\,(x - x_a) + y_a = CA_B\,(x - x_b) + y_b$$

qui devient

$$CA_A\,x - CA_B\,x = CA_A\,x_a - CA_B\,x_b - y_a + y_b$$

en isolant x, on obtient la première équation de

$$\begin{cases} x = \dfrac{CA_A x_a - CA_B x_b - y_a + y_b}{CA_A - CA_B} \\ y = CA_A(x - x_a) + y_a \end{cases}$$

Pour trouver y, on utilise la valeur de x (à présent connue) dans la première équation du système initial.

Nous sommes alors certains que l'arc de cercle commençant en A, finissant en B et passant par X passe sous les points J, K, L, M et N. Il reste à calculer le centre et le rayon de ce cercle.

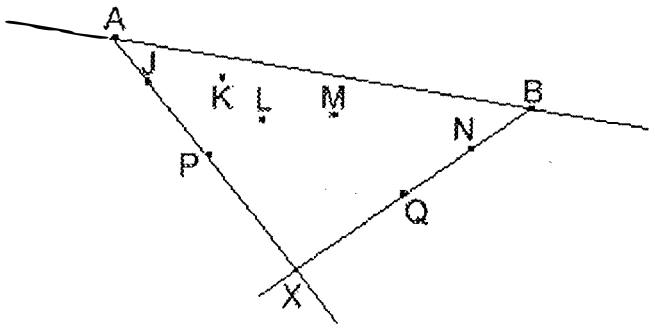


Figure 30: points X, P et Q

Connaissant deux cordes d'un cercle (grâce aux trois points A, B et X), le centre du cercle est l'intersection des médiatrices de ces cordes. Le milieu de [AX] est la moyenne arithmétique des coordonnées des points A et X ; le milieu de BX, celle des coordonnées de B et X.

$$P = \left(\frac{x_a + x_x}{2}, \frac{y_a + y_x}{2} \right)$$

$$Q = \left(\frac{x_b + x_x}{2}, \frac{y_b + y_x}{2} \right)$$

Les droites PC et QC ont pour équation :

$$\begin{cases} PC \equiv y = -\dfrac{1}{CA_A}(x - x_p) + y_p \\ QC \equiv y = -\dfrac{1}{CA_B}(x - x_q) + y_q \end{cases}$$

Rappelons que le coefficient angulaire d'une droite perpendiculaire à une autre est l'opposé de l'inverse du coefficient angulaire de cette seconde droite.

Pour trouver les coordonnées du point C, il faut résoudre le système formé par ces deux équations. Les deux membres de droite étant égaux, le système se transforme en l'équation

$$\frac{-1}{CA_A}(x - x_p) + y_p = \frac{-1}{CA_B}(x - x_q) + y_q$$

$$\frac{-x}{CA_A}+\frac{x}{CA_B}=\frac{x_q}{CA_B}-\frac{x_p}{CA_A}+y_q-y_p$$

$$C\equiv\begin{cases} x=\dfrac{CA_ACA_B}{CA_A-CA_B}\left(\dfrac{x_q}{CA_B}-\dfrac{x_p}{CA_A}+y_q-y_p\right) \\ y=\dfrac{x_p-x}{CA_A}+y_p \end{cases}$$

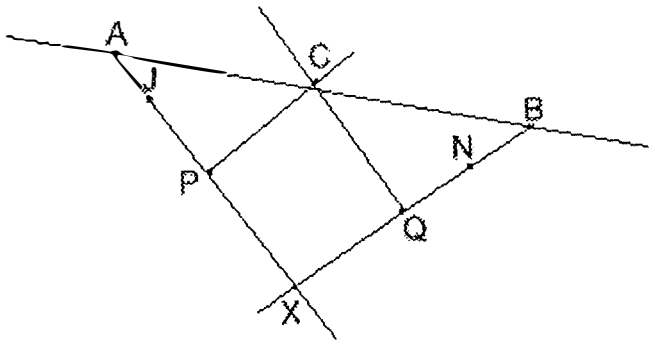


Figure 31: le centre du cercle

Il est nécessaire d'isoler le cas où l'un des coefficients angulaires (CA_A ou CA_B) est nul pour éviter des dépassements de capacité. Ils ne pourraient pas être égaux vu que les droites sont perpendiculaires à deux droites sécantes.

Le rayon du cercle est la distance séparant le centre d'un des points du cercle (soit A) :

$$r=\sqrt{(x_c-x_a)^2+(y_c-y_a)^2}$$

2.5 Écartement des portées

Le lecteur du code fourni en annexe a certainement remarqué que chaque graphique est positionné par rapport au point supérieur gauche de la portée qui le contient et que ce point a pour coordonnée (0, 0) dans la fenêtre (cfr création de la portée graphique dans la méthode Ajoute de CDecoupagePortee). Par ailleurs, il est fait appel, généralement après l'insertion de nouveaux graphiques, à la méthode ChangeDepassements. Nous avons ignoré ces appels jusqu'ici. Il existe des dépassements haut et bas au niveau de la portée de découpage, de la mesure de découpage et des brouillons d'objets graphiques horizontaux. Leur signification a été illustrée à la figure 16 du chapitre précédent.

Change-Dépasse-ments

Au niveau de la portée, ils vont nous permettre de décaler les objets graphiques afin que deux graphiques de portées différentes ne se superposent pas. C'est ce que réalise la méthode EcartePortee de la classe CDecoupagePartition. Les commentaires accompagnant le code devraient suffire à sa compréhension. Mentionnons, malgré tout, que le paramètre *decalage* représente la translation verticale que doivent subir tous les graphiques de cette partition. On peut en effet rencontrer des fenêtres où sont représentées plusieurs partitions (mélodies). Si la première doit bien apparaître à l'ordonnée 0 de la fenêtre, les suivantes doivent être affichées en dessous de celles qui les précèdent.

2.6 Création du fond de couleur

La dernière tâche de la méthode Ajoute de CDecoupagePartition est de créer un fond de couleur (rectangle englobant tous les graphiques de la partition) et de l'ajouter à l'ensemble des graphiques de la partition. Il est ajouté en première position pour apparaître à l'arrière-plan par rapport aux autres graphiques.

3. Autres algorithmes

Ce projet étant un projet-pilote pour Logi+, en ce qui concerne la nature de l'affichage (non alphanumérique) et l'environnement de développement, il nous paraît important de détailler les algorithmes de traitement et de stockage des images (cache).

Les dessins des graphiques musicaux sont stockés, comme ressources, avec le code. Avant de pouvoir être affichés dans une fenêtre, ils doivent subir des modifications coûteuses en temps. Afin d'améliorer les performances lors de l'affichage (éviter l'effet de clignotement lors d'un ré-affichage), il a été décidé de stocker les images (traitées) dans un cache. Nous analyserons, ci-après, la méthode d'initialisation d'un élément du cache, où toutes les transformations sont réalisées, et ensuite, le fonctionnement du cache. Le code correspondant aux deux classes développées ici se trouve en annexe 8.

3.1 CElementCachePixmap

Trois étapes sont nécessaires à la transformation de la ressource en deux pixmaps[23] :

1) Récupération de la ressource.

Cette première étape se fait par un appel à xvt_res_get_image avec le numéro de la ressource désirée en paramètre. Cette procédure nous renvoie une image[24]. Malheureusement, les images ne peuvent s'afficher dans une fenêtre sans effacer tout ce qui se trouve derrière elles. On se trouve donc dans l'obligation de passer par des pixmaps, objets que l'on ne peut stocker en ressource mais qu'il est possible de superposer à d'autres dessins affichés précédemment dans la fenêtre. Les deux autres étapes concernent la création du pixmap noir-et-blanc et du pixmap couleur.

2) Création et initialisation du pixmap noir-et-blanc.

On crée le pixmap par xvt_pmap_create, en fournissant la taille désirée. On l'initialise en dessinant l'image dans un cadre du pixmap. Le mécanisme est similaire à celui utilisé pour afficher un pixmap dans une fenêtre (cfr figure 17 et 32).

---

[23] le fait que deux pixmaps soient nécessaires a été expliqué dans la section concernant l'affichage.

[24] les termes *image* et *pixmap* utilisés ici font référence à des types définis par xvt : XVT_IMAGE et XVT_PIXMAP.

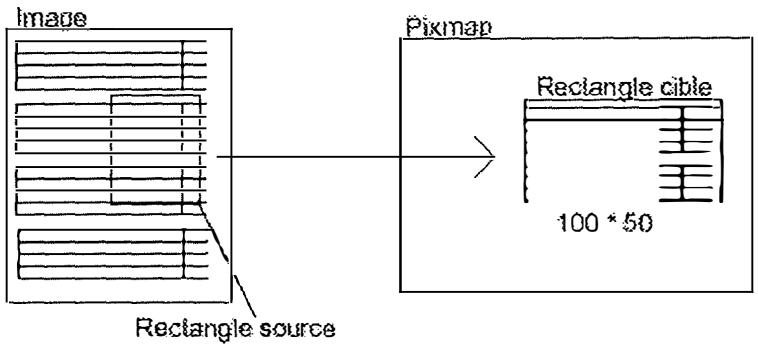


Figure 32: initialisation du pixmap

3) Création et initialisation du pixmap couleur

Le pixmap couleur est identique au pixmap noir-et-blanc dont on a remplacé le noir par une autre couleur. On va donc modifier la palette de l'image en remplaçant la couleur d'indice 0 par la couleur désirée (lors de la création des ressources, on a prévu que l'indice 0 de la palette de couleurs est toujours le noir). Ensuite, la création et l'initialisation du pixmap sont semblables à celles du pixmap noir-et-blanc et utilisent les mêmes paramètres.

3.2 CCachePixmap

Les éléments se retrouvent dans le cache dans un vecteur ordonné, du plus récemment accédé au moins récemment accédé. Chaque élément est identifié par son numéro de ressource, la couleur de son élément coloré et l'échelle à laquelle il a été dimensionné.

Nous avons développé deux algorithmes de cache. Le premier était idéal, à cache de taille fixe, il est irréalisable en utilisant les procédures offertes par xvt pour des raisons que nous expliquerons plus loin. L'autre algorithme, à cache de taille variable est, lui, réalisable avec les procédures xvt. Commençons par expliquer l'algorithme idéal.

Lorsqu'on désire accéder à un pixmap, on le cherche dans le vecteur des éléments. Si on le trouve, on l'en retire et on le réinsère en première position. Sinon, on le crée et on l'insère en première position dans le vecteur dont on a enlevé le dernier élément s'il a atteint sa taille maximum. L'élément demandé est finalement renvoyé au requérant.

xvt ne permet pas d'implémenter cette solution élégante. En effet, xvt empêche la destruction de pixmaps lors d'un affichage ou de la mise à jour du contenu d'une fenêtre (seuls moments où le cache est utilisé !).

L'autre solution consiste à ne pas vider le cache au fur et à mesure des accès mais de le faire par un appel à une méthode VideCache appelée après l'affichage ou la mise à jour de la fenêtre. Cette solution, plus coûteuse en mémoire, est plus efficace car il n'est nécessaire de créer un pixmap qu'une seule fois par fenêtre. L'appel à VideCache devant se faire dans une méthode de la fenêtre (en réponse à un événement d'affichage ou de mise à jour), cette solution est aussi moins transparente que la première.

Chapitre 5. Implémentation en langage Java

1. Java

Faut-il encore présenter le langage Java ? L'inclusion d'applets (petites applications Java) dans les pages HTML a donné à ces pages une nouvelle dimension dans l'interactivité et a fait connaître Java du grand public. Sa popularité dans le milieu informatique est telle que peu de monde peut se targuer de n'en avoir jamais entendu parler.

Au-delà d'être ce « gadget » embellissant Internet, Java est un langage de programmation (inspiré du C++) orienté-objet, permettant l'héritage (simple et multiple), l'encapsulation, le polymorphisme, le multi-threading, les liens dynamiques. Il a tout pour être utilisé à grande échelle dans l'industrie.

Le langage Java se démarque du C++ par au moins deux aspects importants : sa portabilité et son ramasse-miettes (garbage collector).

Le code source Java est compilé en ce qui est communément appelé le byte code Java. Un interpréteur de byte code est utilisé pour produire le résultat de l'application. Le byte code Java est donc portable sur n'importe quelle machine possédant un interpréteur Java. « *This allows developers to hire programmers based on the programmer's expertise with the type of applications they want to write, not just the kind of machine they know to use* »[25].

Le ramasse-miettes évite aux programmeurs de s'occuper de la libération de la mémoire allouée dynamiquement. Le remplacement des pointeurs par des références rend obsolètes les opérations sur pointeurs, sources de nombreuses erreurs en C++.

Le langage Java étant un langage jeune, des améliorations sont prévisibles. Avec le développement de compilateurs « Just In Time », convertissant « on the fly » le byte code en code natif, les applications Java devraient atteindre les performances des applications écrites spécialement pour chaque plate-forme. Les outils d'aide au développe-

[25] Bill Kelly, président d'Ignite in [HALF,97]

ment Java s'amélioreront et permettront aux programmeurs d'atteindre la productivité qu'ils possèdent avec les outils dédicacés aux autres langages.

2. Traduction C++ - Java

Traduire du code C++ en Java est fort mécanique. Les différences entre ces deux langages sont surtout d'ordre syntaxique. De plus, les classes graphiques de Java offrent quasi les mêmes services que les procédures xvt utilisées pour le projet C++. Cette section ne reprendra pas toutes les différences syntaxiques qui existent entre C++ et Java, ni la comparaison des services offerts par xvt et ceux offerts par les classes graphiques de Java. Elle expliquera plutôt les différences de fond existant entre l'afficheur développé en C++ et sa version Java : la position du champ musical dans la présentation du résultat de la recherche, le mécanisme de restauration des ressources graphiques (dessins) et l'objet graphique de liaison (l'objet « arc de cercle » est le seul où des différences importantes existent entre xvt et Java).

2.1 Position du champ musical

La version Internet de l'afficheur se veut la plus ressemblante possible de la version Windows ou Macintosh en ce qui concerne les résultats. L'utilisateur a[26] donc la possibilité de définir ses paramètres de présentation des données par le navigateur Internet.

La partie « présentation » relevant moins du projet musical que du projet général, nous ne l'avons pas analysée et ne la décrirons pas en détail. Expliqué simplement, l'afficheur range les zones de texte et de musique dans un tableau dont il calcule les dimensions des colonnes au mieux. Dans la version Internet, il s'agit d'un script CGI qui consulte la base de données, en extrait les informations pertinentes en regard de la requête de l'utilisateur et transforme ces résultats en une page au format HTML qu'il envoie à l'adresse de l'utilisateur. C'est le navigateur, donc du côté du client, qui se charge de la césure des phrases et du calcul des dimensions des colonnes pour que la présentation du résultat soit optimale.

Au début du projet « Afficheur musical sur Internet », une solution évitant la traduction en Java avait été imaginée. Le CGI insérait dans la page HTML, à l'emplacement du champ musical, un appel à un autre CGI, qui renverrait une image (au format IMG d'HTML). Ce CGI serait presque la réplique de l'afficheur musical C++. En effet, celui-ci affiche des pixmaps dans une fenêtre ; il aurait été très facile de le transformer pour copier le contenu de cette fenêtre dans une image et de renvoyer cette image en un flot de bits. Cette solution nous aurait fait gagner un temps considérable et son résultat aurait été identique à celui de la version C++. Un problème majeur est cependant venu l'exclure : le serveur Internet sur lequel est implémenté le script général est un serveur 32 bits et Logi+ ne possède une licence xvt que pour des applications 16 bits. La repro-

[26] ou plutôt aura car la partie « présentation » sur Internet est en cours d'implémentation.

grammation des procédures d'xvt, tout comme la commande de la version 32 bits d'xvt, étant exclue, il a été décidé d'abandonner l'idée de CGI pour s'investir dans la traduction du code C++ en Java.

L'inclusion d'une applet Java dans un tableau (le tableau présentant les résultats de la recherche) ne présente pas de problème mais il faut en connaître les dimensions exactes avant son exécution. Les dimensions de la partition n'étant connues qu'après son découpage, cette solution simple était exclue. Deux solutions de remplacement ont été étudiées.

La première consiste à définir une taille constante pour l'applet et de prévoir un ascenseur vertical pour le cas où la partition est plus grande que l'emplacement prévu. Mais de la place serait perdue dans le cas où la mélodie est plus petite que l'emplacement prévu. De plus, le temps d'attente avant que ne s'affiche la mélodie serait très long vu que le découpage de toutes les mélodies se déroulerait parallèlement.

La seconde solution a été retenue. Seul un bouton est affiché dans le tableau. Il a une taille fixe et déterminée. Lorsque l'utilisateur appuie sur ce bouton, le découpage de la partition associée à la fiche contenant le bouton est lancé et la mélodie s'affiche dans une fenêtre indépendante du navigateur.

Même si elle s'éloigne de la solution C++, la solution « bouton » a été adoptée car

- le redimensionnement de la fenêtre est possible : elle englobe donc l'entièreté des mélodies affichées[27] ;
- elle permet à l'utilisateur de présenter une fiche valablement (en modifiant l'emplacement de la fenêtre) ;
- l'ajout d'un menu permettant l'accès à des informations de type « aide » a été possible (l'aide se limite, pour le moment, à l'explication des panneaux d'avertissement) ;
- un bouton similaire pourra être placé pour « jouer » la mélodie ;
- seuls l'initialisation et l'affichage des boutons sont effectués en parallèle lors de l'affichage de la page HTML ; les opérations de découpage et d'affichage ne sont réalisées que lorsque l'utilisateur le désire.

Un seul regret : l'utilisateur ne voit pas la mélodie lorsqu'il imprime le résultat de sa recherche. Mais, si la première solution avait été adoptée, il n'aurait vu le plus souvent qu'une partie de la mélodie.

2.2 Ressources graphiques

Nous avons vu dans le chapitre précédent que les dessins des graphiques musicaux sont stockés, comme ressources, avec le code compilé. En Java, l'applet les connaît par un autre moyen. Les dessins sont stockés sur le serveur. C'est la classe CRessources qui se

[27] un bug tenace fait que cela n'est pas encore le cas sur toutes les plates-formes.

charge de les charger en mémoire de la station-client. Cette classe possède deux méthodes publiques (cfr annexe 9).

La première est appelée lors du découpage de la mélodie : lorsqu'on crée un graphique contenant un dessin, ChargeImage est invoquée avec le numéro d'identification du dessin en paramètre. Si le chargement de ce dessin n'a pas encore débuté, cette méthode enclenche ce chargement.

Lorsqu'une image doit être affichée (lors de la phase d'affichage), RetrouveImage est invoquée. Cette méthode vérifie que toutes les images dont on a demandé le chargement ont bien été chargées. Ensuite, elle renvoie l'image demandée.

Cette classe permet de gagner le temps de chargement des dessins ; en effet, ils sont chargés alors que le découpage s'effectue.

Données contenues	
Image cache[] ;	Liste des images chargées ou en cours de chargement.
static String urlImages[] ;	Tableau reprenant pour chaque image, le nom du fichier la contenant sur le serveur.
MediaTracker pion ;	Objet chargé de la surveillance du chargement des images.
Applet appletMere ;	Référence de l'applet requérante.
boolean chargementTermine ;	Etat du chargement des images du cache.
Services offerts à tous	
CRessources (Applet appletAppelante)	Constructeur de l'objet.
void ChargeImage(int *numeroRessource*);	{Post-condition} Le chargement de l'image identifiée par *numeroRessource* a débuté.
Image RetrouveImage(int *numeroRessource*);	{Post-condition} Sauf erreur, le chargement de toutes les images est terminé, l'image demandée est renvoyée. Si une erreur est survenue, elle a été signalée à l'utilisateur.

2.3 Objet graphique de liaison

En Java, un arc d'ellipse est défini par un rectangle déterminant la taille de l'ellipse (comme en xvt) et par deux angles (cfr figure 33). Le premier détermine le point de début de l'arc ; le second la grandeur de l'arc (les angles positifs déterminent une rotation dans le sens contraire au sens des aiguilles d'une montre).

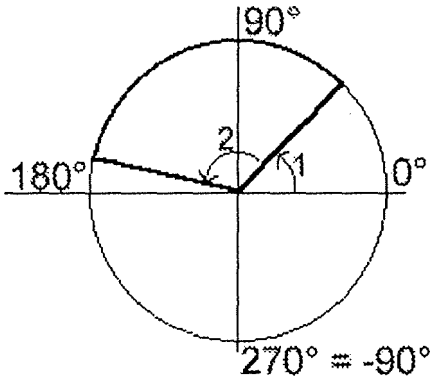


Figure 33: angles d'un arc de cercle

En conséquence, une nouvelle méthode privée a été ajoutée à la classe CBrouillonGraphiqueLiaison : CalculeAngles (cfr annexe 10) qui, à partir des coordonnées des points de début et de fin de la liaison et du cercle, calcule les deux angles décrits ci-dessus.

Chapitre 6. Améliorations possibles

Tout programme, tout projet est une étape. Il est le fruit de recherches, de travaux, mais doit aussi susciter de nouveaux projets. Ce chapitre se veut l'ébauche d'une nouvelle recherche de « requirements », en vue d'une nouvelle version de l'afficheur. Il est né de choix de conception, qui seront justifiés, d'oublis dans les spécifications, d'erreurs de conceptions. La dernière section est consacrée aux améliorations concernant la version Java.

1. Spécifications

A la fin de mon stage, l'afficheur a été testé par quelques utilisateurs d'InfoMusic. Les deux sections suivantes proviennent de leurs réflexions, de leurs suggestions.

1.1 Tempo

Dans cette version du logiciel, le tempo apparaît toujours sous la forme ♩= x. Or, la théorie musicale nous apprend que les indications de tempo doivent se référer à la durée d'un temps de la mesure. Le programme devrait donc, en fonction de la mesure, afficher d'autres indications telles celles incluses dans le tableau ci-dessous.

Mesures binaires	$\frac{4}{1}$: 𝅝 = X	$\frac{4}{2}$: 𝅗𝅥 = X	$\frac{4}{4}$: ♩ = X	$\frac{4}{8}$: ♪ = X
Mersures ternaires	$\frac{6}{1}$: 𝅝. = X	$\frac{6}{2}$: 𝅗𝅥. = X	$\frac{6}{4}$: ♩. = X	$\frac{6}{8}$: ♪. = X

Cette correction demandant des changements conséquents, cet oubli sera corrigé lors d'une version ultérieure de l'afficheur. En effet, en plus d'un nouveau code (d'indication de tempo), il aurait fallu soit postposer la création de l'indication de tempo à la connaissance de l'indication de mesure, ce qui est impossible vu que l'indication de mesure n'est pas obligatoire, soit définir une règle dans le nouveau codage obligeant l'indication du tempo à suivre l'indication de mesure (si elle existe). Dès lors, l'analyse du tempo et l'ajout des graphiques musicaux qui s'y rapportent ne se feraient plus au niveau de la partition de découpage mais au niveau de la mesure de découpage.

Je suggère que lorsque cette modification sera apportée, une autre soit directement réalisée : le tempo n'étant plus encapsulé dans le paquet d'en-tête, l'implémentation des changements de tempo en cours de mélodie serait facilement réalisable. Cette possibilité, quoiqu'offerte par la théorie musicale, n'a pas été jugée utile aux utilisateurs. L'avenir nous apprendra s'ils en feront la demande.

1.2 Points

Les points (qui se trouvent à droite de la note et modifient sa durée) sont toujours dessinés à mi-hauteur du corps de la note. Cette situation fait que, lorsque la note est placée sur une ligne de la portée, il est parfois difficile de distinguer le point de la ligne. La théorie musicale nous indique que, lorsque la note pointée se trouve sur une ligne, le point se dessine dans un interligne voisin de la ligne où est dessinée la note (il s'agit généralement de l'interligne du dessus).

2. Conception

2.1 Graphiques musicaux

Au début de la phase de conception, deux possibilités d'affichage s'offraient à moi. La première est celle qui a été adoptée : créer des dessins de tous les graphiques musicaux, les modifier à la guise de l'utilisateur et les afficher à l'écran. Les performances de cette solution laissant à désirer, la création du cache de pixmaps (cfr chap. 4, section 3) a été décidée. Je pense que la seconde solution serait plus performante. Elle consiste en la création d'une police de caractères vectorielle où sont dessinés tous les graphiques utiles. Cette solution possède deux atouts majeurs :

1) Le redimensionnement des graphiques se fait rapidement et presque sans perte. Dans la version actuelle, les graphiques sont optimisés pour une réduction à l'échelle ½. Si l'utilisateur désire un rapport différent, la réduction de certains graphiques résultera en des pertes, parfois importantes, de lisibilité (disparition de lignes horizontales et/ou verticales).

2) L'affichage des graphiques se fait plus rapidement car cette solution rend inutile le double affichage (le fond d'un texte est facilement défini « transparent ») et l'affichage de texte est nettement plus optimisé dans xvt que celui des images.

Malheureusement, cette solution présente un inconvénient de taille : le développement de la police vectorielle aurait pris au moins un mois de travail avec les outils dont nous disposions. Pour cette raison, nous avons adopté la première solution, qui, même si elle est nettement moins efficace, permettait d'espérer conclure ce projet en quatre mois.

Pour une version future, je pense qu'il serait utile de tester les performances de la seconde solution et, si ces tests sont concluants, de créer cette police de caractères ou de négocier des droits d'utilisation de polices musicales existantes (il en existe chez Adobe,

par exemple). Ce changement aura peu de répercussions au niveau des classes d'affichage car l'affichage d'images est encapsulé dans la classe mère CGraphiqueMusical. Le cache de pixmaps ne sera plus nécessaire.

2.2 Stockage des objets graphiques

Lors du découpage, les objets graphiques sont stockés au niveau de la partition (CPartition) et au niveau de la mesure de découpage (CDecoupageMesure). Lors de la création d'un graphique, sa référence est notée aux deux niveaux. Lorsqu'il faut supprimer les graphiques d'une mesure, la situation est assez confuse (cfr méthode EffaceObjetsGraphiques de CDecoupageMesure).

Je trouve qu'il serait plus élégant de stocker, pendant le découpage, les graphiques musicaux au niveau des mesures de découpage. En fin de découpage, les différentes structures de données contenant les graphiques seraient concaténées en une seule dépendant de la partition. Il se fait que je n'y ai pas pensé lors de la conception. La correction de cet oubli implique, outre l'implémentation de la phase de regroupement des ensembles de graphiques et la suppression de la méthode EffaceObjetsGraphiques, un travail de scribe consistant à supprimer toutes les insertions de graphiques dans l'ensemble se trouvant au niveau de la partition de découpage.

2.3 Informations de découpage dans les objets graphiques

Il avait été décidé, au début de la phase de conception, que les objets graphiques musicaux ne contiendraient que des informations utiles à leur affichage. Cependant, il a été très pratique d'y encapsuler des informations utiles au découpage, comme le fait que le graphique représente une note, un silence, qu'il puisse terminer une mesure ou, plus flagrant, que le graphique connaisse le décalage que peuvent subir les graphiques à sa droite (décalage idéal) lors de la justification des portées.

S'il s'avère que le programme ne peut fonctionner à cause d'un manque de mémoire lors de l'affichage ou si un bon programmeur maniaque (ne serait-ce pas un pléonasme ?) désirait améliorer cette situation peu élégante, une solution serait de créer des brouillons d'objets graphiques qui ne seraient transformés en objets graphiques musicaux qu'en fin de découpage. Je pense néanmoins que la situation actuelle est un bon compromis entre élégance, performance et gestion de mémoire.

3. Amélioration de l'applet Java

3.1 Affichage en couleur

Il est prévu d'afficher les graphiques musicaux dans la couleur demandée par l'utilisateur, comme c'est le cas dans la version C++. Java offre cette possibilité : le code de changement de couleur se trouve dans les classes CCacheImages, CElementCacheImages et CFiltreCouleur. Il n'est pas utilisé dans la version actuelle. En effet, des tests

ont montré que les images colorées, lorsqu'elles sont affichées, se mettent à clignoter (comme si elles étaient réaffichées continuellement). Le problème isolé, il est apparu qu'il provient d'une des classes graphiques de Java. Il faut donc attendre une version future du Java Development Kit et réessayer, en espérant que ce bug soit corrigé.

3.2 Chargement des classes

Une des améliorations nécessaires au bon fonctionnement de l'applet d'affichage de la musique est l'accélération du chargement des classes.

Dans la version actuelle, le byte code Java est chargé par le classLoader du navigateur Internet. Pendant l'interprétation du code, lorsqu'une classe est nécessaire, le classLoader établit une connexion avec le serveur, transfère le fichier contenant le byte code de la classe demandée et clôt la connexion. Puisque c'est au cours de la phase de découpage que la majorité des classes sont chargées, le chargement de quelque 50 classes (au pire 90 !) est nécessaire avant que la partition ne soit affichée. Cette situation, conjuguée à l'état d'encombrement du réseau Internet, impose à l'utilisateur une attente intolérable. La dernière semaine de mon stage a été consacrée à la recherche de solutions à ce problème.

Une solution serait de faire patienter l'utilisateur en lui présentant une barre de progression indiquant l'avancement du découpage. Je pense que si elle a le mérite d'offrir un feedback à l'utilisateur, cette solution ne règle pas le problème de fond : la lenteur du découpage.

Java permet de redéfinir la classe ClassLoader. Il serait donc possible de lui faire charger les classes par paquets. En effet, ce n'est pas le temps de transfert qui ralentit le découpage, mais bien le temps de connexion et de déconnexion au serveur. Cette solution permettrait certainement de diminuer le temps de chargement des classes d'un facteur 5. Le manque de temps et la complexité de la définition d'un ClassLoader font rapidement oublier cette solution.

La réponse est venue de la nouvelle version de Java (1.1). A l'époque, une version bêta était déjà en circulation et nous avons pu la tester. La nouvelle version prévoit que les classes peuvent être regroupées (et même compactées) dans des fichiers (dont l'extension sera .JAR). Il suffit de fournir, en paramètre de l'applet, les noms des fichiers où se trouvent les classes afin que le ClassLoader « nouvelle version » les retrouve (après avoir chargé ces fichiers et les avoir décompactés). Naturellement, les navigateurs Internet ne sont pas encore équipés de ce nouveau ClassLoader mais, après la sortie de Java version 1.1, cela ne devrait pas tarder. Les fichiers regroupant les classes ont déjà été créés et le script CGI est déjà modifié pour inclure ces fichiers aux paramètres de l'applet.

Actuellement, le chargement des classes est encore extrêmement lent mais une solution a été mise en oeuvre de sorte que, dans un avenir proche, les performances s'améliorent de manière spectaculaire.

Conclusion

Ce travail, même s'il n'apporte pas une grande contribution à la recherche informatique, a, sans nul doute, contribué au développement d'un de ses domaines quasi non marchand : la musique. En effet, les associations musicales (conservatoires, académies de musique, centres de musique, ...) ne reçoivent généralement qu'une petite part du budget (déjà maigre) alloué à la culture ; elles n'ont, de ce fait, pas les moyens de financer de grands projets liés à l'informatique. Or, les chargés de cours et les professeurs des grands instituts belges de musicologie (IMEP en Wallonie, Lemmens Instituut en Flandre) ne manquent pas d'idées, ni de projets d'ordre pédagogique (enseignement de la musique assisté par ordinateur), logistique (gestionnaire de base de données musicales) ou concernant des points de recherche particuliers (simulation de différents tempéraments[28] d'instruments à clavier). La situation de Namur, qui se veut une ville culturelle et la « capitale du chant choral », les institutions culturelles et musicales qui y sont implantées, devraient inciter notre Institut d'Informatique à encourager la recherche dans ces domaines et à organiser des synergies avec l'Imep, le CIMC. Si ce doux rêve de voir des informaticiens se lancer dans de tels projets devient réalité, je pense que le chapitre premier de ce travail devrait constituer, pour eux, une bonne introduction à la musique.

Pour l'entreprise Logi+, les deux premiers chapitres permettent à un nouveau membre d'équipe (peut-être stagiaire) de se familiariser aisément avec le projet InfoMusic. Les deux chapitres suivants sont davantage destinés à la maintenance du projet. Grâce au chapitre 3 et à sa liste exhaustive de classes et de services offerts, il est facile de repérer l'endroit où une modification est nécessaire et où un nouveau service trouve sa place. Les subtilités du code expliquées au chapitre 4 permettront à un informaticien de comprendre les algorithmes utilisés et de les modifier, si nécessaire. Enfin, l'expérience acquise en Java, suite à ce projet, donne à Logi+ un nouvel avantage sur le plan des solutions télématiques. Avantage qui pourrait amener la Communauté Européenne à subsidier, à l'avenir, le projet Musica (et le projet informatique qui lui est associé).

---

[28] Le tempérament est la manière de répartir les intervalles de la gamme sur un clavier. Les plus connus sont les tempéraments égal (le plus usité), pythagoricien et zarlinien.

Si l'évaluation du produit est assez positive, l'expérience que j'ai retirée de mon stage et de la rédaction de ce mémoire a été considérable.

Ces travaux m'ont permis, entre autres choses, de mettre en pratique les connaissances acquises durant les quatre années précédentes. Par exemple, les concepts introduits aux cours de théorie des langages et de programmation ont été utilisés dans l'approche des langages C, C++ et Java qui, avant juillet 1996, m'étaient totalement inconnus. J'ai également appliqué les techniques de gestion de projets et de planification, introduites au cours, pour gérer la durée des phases de développement et évaluer les conséquences des choix d'implémentation en terme de travail supplémentaire.

La pratique de la programmation à Logi+ m'a offert une très bonne maîtrise de la programmation orienté-objet. En effet, suite au choix de C++ comme langage de programmation, la stratégie de Logi+ a été d'utiliser le paradigme de programmation orienté-objet et d'en exploiter les concepts au maximum. Cela implique des efforts : encapsuler correctement les données, vérifier que chaque variable membre trouve bien sa place dans la classe où elle est encapsulée, scinder certaines classes lorsqu'elles englobent trop de concepts ou que leur sémantique est trop difficile à saisir. Ces efforts sont souvent récompensés par un code clair, facile à comprendre.

Certes, les cours théoriques et les travaux académiques avaient introduit correctement tous les concepts nécessaires à la compréhension de ce paradigme. Mais l'ampleur du travail réalisé en quatre mois et le suivi dont mon travail a été l'objet de la part de Monsieur Éric Bishoff, mon superviseur de Logi+, m'ont permis d'intégrer tous ces concepts en les utilisant régulièrement et de façon intensive.

Ma capacité à évaluer la complexité et le temps nécessaire à la réalisation d'un projet s'est affinée. Une expérience sur un projet grandeur nature et sur lequel on travaille à temps plein, donne une autre vue des projets que les problèmes théoriques offerts tout au long de son cursus à l'étudiant en informatique. Le fait de travailler à un projet de manière discontinue (en suivant des cours) cache un paramètre de grande importance dans la gestion du temps : la productivité. Et c'est seulement en connaissant sa productivité qu'il est possible d'évaluer le temps qu'une tâche requerra.

Durant la réalisation de ce travail j'ai expérimenté combien la documentation est importante. Lors de la rédaction du chapitre 4 (six mois après la clôture du projet), les commentaires annexés au code m'ont été très utiles ; mais je me suis rendu compte qu'ils ne suffisent pas et que des documents semblables aux chapitres 3 et 4 seraient de la plus grande utilité au niveau de tous les projets de Logi+.

Quelques difficultés ont émaillé ce travail. La première, et non la moindre, fut d'« entrer » dans le projet : comprendre ce que l'on désirait de moi, le fonctionnement des modules connexes et leurs relations avec l'afficheur. Ensuite, durant la phase de conception, gérer la complexité du projet et planifier ses différentes étapes afin de terminer dans les délais ont été des préoccupations quasi quotidiennes. Lorsque la réalisation de l'afficheur musical a été achevée,

l'intégration dans le projet principal a donné lieu à d'autres problèmes, souvent liés au manque de documentation au niveau des classes (sémantique, durée de vie...).

Durant la traduction en Java, la jeunesse du langage a constitué la difficulté principale : je ne disposais pas d'un livre de référence digne de ce nom et la documentation fournie sur le site de Sun comportait des lacunes importantes (par exemple, le mode de passage des paramètres n'y était mentionné nulle part !).

Fin janvier, la version C++ de l'afficheur avait été testée et se trouvait dans un état propre (sans bug connu) ; la version Java était installée sur le serveur Musica. Je pense donc avoir correctement surmonté ces problèmes, avec l'aide constante d'Éric Bishoff.

J'espère que ce mémoire vous a plu dans son fond et dans sa forme et que vous avez pris autant de plaisir que moi à associer l'informatique à l'harmonie de la musique.

Bibliographie

[DRIX,93] : Philippe DRIX, *Langage C norme ANSI vers une approche orientée objet*, Masson, 1993

[HALF,97] : Tom R. HALFHILL, *Today the Web, tomorrow the World*, in *Byte*, Janvier 1997, pp. 68-80

[HECK,95] : Jim HECKROTH, *Tutorial on MIDI and Music Synthesis*, http:// www. harmony-central. com/ MIDI/ Doc/tutorial.html (dernier hit : août 1997)

[HIND,86] : Paul HINDEMITH, *Pratique élémentaire de la musique*, traduit et annoté par Robert Mermoud, Ed. Jean-Claude Lattès, 1986.

[LIPS,89] : Éric LIPSCOMB, *Introduction into MIDI*, http :// www. harmony-central. com/ MIDI/ Doc/ intro.html (dernier hit : août 1997)

[MACH,52] : Armand MACHABEY, *La notation musicale*, Coll. Que sais-je ?, Presses Universitaires de France, 1952

[MCQU,95] : Bob McQUEER, *The USENET MIDI Primer*, http :// harmony-central.mit.edu/ MIDI/ Doc/ primer.html (dernier hit : avril 1996)

[NEWM,96] : Alexander NEWMAN & al., *Le Macmillan Java*, S&SM, 1996

[PIER,87] : John R. PIERCE, *Le son musicalMusique, acoustique et informatique*, L'Univers des science, Pour la Science Diffusion Belin, 1987

[SMF,96] : *The MIDI File Format*, http ://www.servtech.com/~jglatt/tech/midifile.htm (dernier hit : mai 1996)

[VIGN,87] : *Dictionnaire de la Musique*, sous la direction de Marc Vignal, Librairie Larousse, 1987

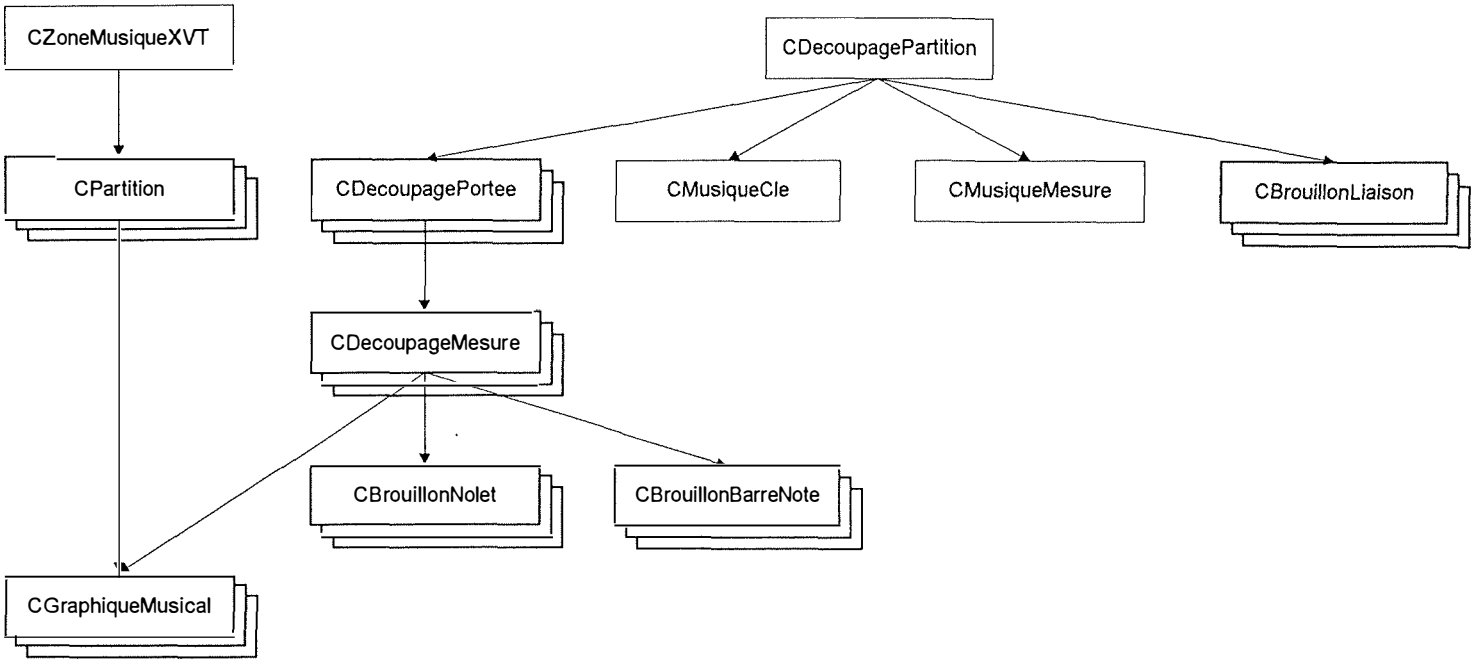
[XVT1,94] : *XVT Portability ToolKit reference, release 4.0*, XVT, 1994

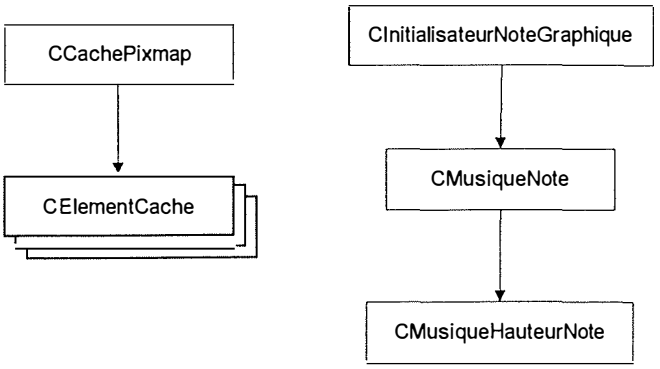
[XVT2,94] : *Guide to XVT development solutions for C*, XVT, 1994

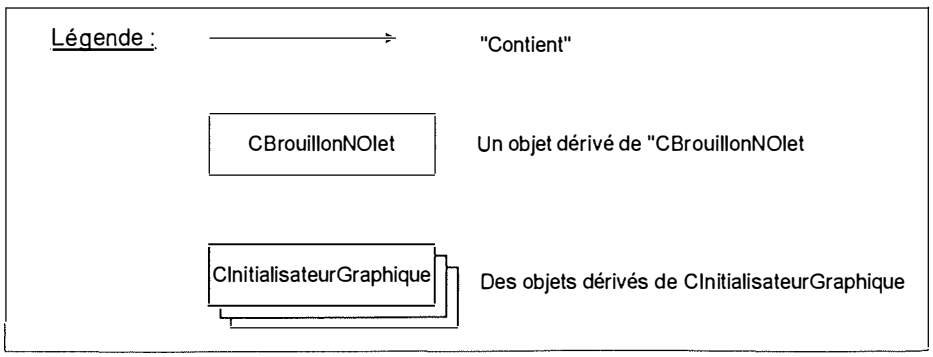
Annexe 1 : Relation entre classes

Avertissement : les deux premiers graphes de cette annexe peuvent être incorrectement interprétés. Il est utile de se reporter aux explications fournies au début du chapitre 3.

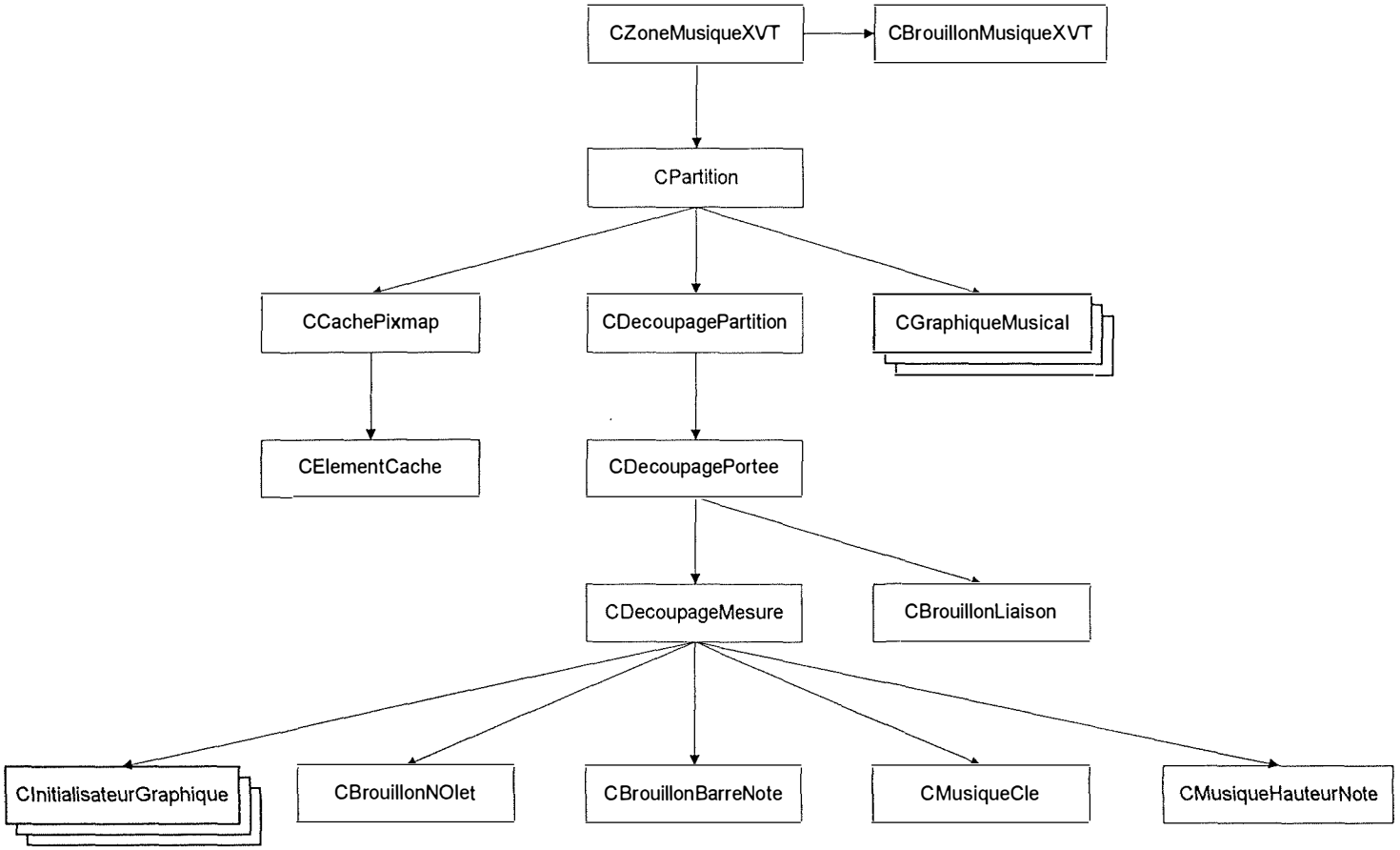
Relation "contient"

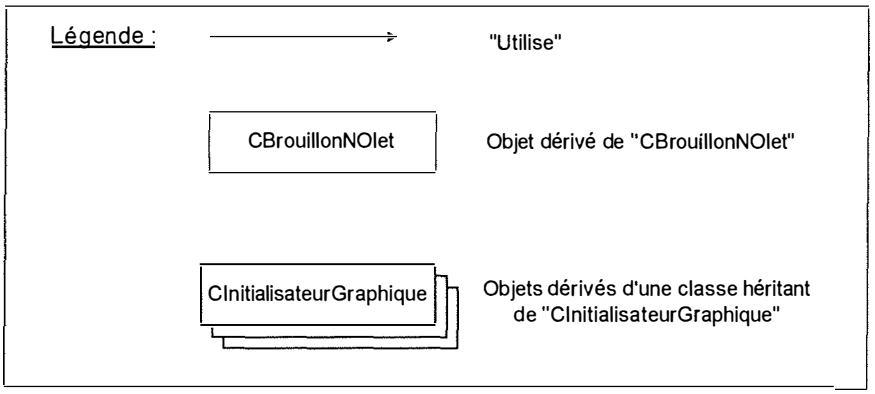






Relation de sous-traitance





Hiérarchie des classes

CCachePixmap

CMusiqueCle

CMusiqueMesure

CMusiqueHauteur-Note

CMusiqueNote

CBrouillonXVT

CZoneXVT

CInitialisateur-Graphique

CBrouillonTexteXVT

CBrouillonMusique-XVT

CZoneTexteXVT

CZoneMusiqueXVT

CInitialisateur-GraphiqueRythme

CInitialisateur-AlterationGraphique

CInitialisateurSilence-Graphique

CInitialisateurNote-Graphique

CObjet

CGraphiqueMusical

CPartition

CDecoupageMusical

CElementPixmap

CBrouillonGraphique-Horizontal

CDecoupagePartition

CDecoupageMesure

CDecoupagePortee

CBrouillonGraphique-Liaison

CBrouillonGraphique-Barre

CBrouillonGraphique-NOlet

CGraphiqueDiese

CGraphiqueBemol

CGraphiqueBecarre

CGraphique-DoubleDiese

CGraphique-DoubleBemol

CGraphique-CercleRouge

CGraphique-IconeMesure

CGraphique-IconeArmature

CGraphique-IconeRythme

CGraphiqueFond

CGraphique-BarreDeNote

CGraphiqueLiaison

CGraphiqueNOlet

CGraphiqueCleSol

CGraphiqueCleFa

CGraphiqueCleUt

CGraphiqueMesure

CGraphiqueMesureC

CGraphique-BarreDeMesure

CGraphique-Continuateur-DeMesure

CGraphiquePortee

CGraphiqueTempo

CGraphiquePointe

CGraphique-MesureResumee

CGraphique-MesureCBarre

CGraphiqueNote

CGraphiqueSilence

CGraphiqueCarree

CGraphique-NoteHampe

CGraphiqueRonde

CGraphique-DoublePause

CGraphiquePause

CGraphique-DemiPause

CGraphiqueSoupir

CGraphiqueBlanche

CGraphique-NoteMoustachue

CGraphique-DemiSoupir

CGraphique-QuartSoupir

CGraphique-8iemeSoupir

CGraphique-16iemeSoupir

CGraphiqueNoire

CGraphique-32iemeSoupir

CGraphique-64iemeSoupir

Légende :

"est la classe mère de"

CBrouillonNOlet

Classe "CBrouillonNOlet"

Annexe 2 : CGraphiqueMusical

CGraMusi.h

```
// Nature: Définition de la classe décrivant un objet graphique musical
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 07/10/96
// Modifications:
// Commentaires:

#ifndef CGraphiqueMusical_H
#define CGraphiqueMusical_H

#include "xvt.h"

#include "WaveDef.h"
#include CObjet_i

class CPartition;

class CGraphiqueMusical : public CObjet
{     private:                        // Fonctions à usage privé:
            void AffichePolygone      //    Affichage d'une ligne dans une
                                            fenêtre par l'affichage d'un
                                            polygone
                  (PNT *, PNT *, CPEN *, WINDOW) const;

      protected:                      // Implémentation en mémoire:
            long positionX,           //    Position de l'objet relative à la
                                            fenêtre
                 positionY;

                                      // Fonctions à usage protégé:
            void Position             //    Donne la position de l'objet
                                            graphique musical relative à la
                                            fenêtre visible
                  (long, long, PNT &) const;
            void AfficheImage         //    Affichage de l'image de l'objet à
                                            la coordonnée de l'objet
                  (int, int, int, short, short,
                   long, long, WINDOW, const CPartition *) const;
            void AfficheLigne         //    Affichage d'une ligne dans l'objet
                  (const PNT *, const PNT *, CPEN *,
                   long, long, WINDOW, const CPartition *) const;
            void AfficheNombre        //    Affichage d'un nombre dans
                                            l'objet
                  (short, const XVT_FNTID, short, short,
                   long, long, WINDOW, const CPartition *) const;

      public:                         // Interface standard:
            CGraphiqueMusical         //    Constructeur
                  (long, long);       //    Coordonnées de l'objet graphique
            virtual void Affiche      //    Affichage de la musique
                  (long, long, const CPartition *, WINDOW) const = 0;
            virtual BOOLEAN Note()    //    L'objet est-il un objet graphique
                                            de note ?
                 const {return FALSE;}
            virtual BOOLEAN Silence()
                                      //    L'objet est-il un objet graphique
                                            de silence ?
                 const {return FALSE;}
```

```
            virtual BOOLEAN         //    L'objet permet-il de cloturer une
                                          mesure ?
                  GraphiqueFinMesure() const {return FALSE;}
            virtual int DecalageIdeal
                                    //    Décalage proportionnel
                  () const {return 0;}
            inline long PositionX() //    Accès à l'abscisse de l'objet
                  const {return positionX;}
            inline long PositionY() //    Accès à l'ordonnée de l'objet
                  const {return positionY;}
            inline void PositionX   //    Changement de l'abscisse de
                                          l'objet
                  (long nouvelleValeur) {positionX = nouvelleValeur;}
            inline void PositionY   //    Changement de l'ordonnée de
                                          l'objet
                  (long nouvelleValeur) {positionY = nouvelleValeur;}
};
#endif
```

CGraMusi.cpp

```
// Nature: Implémentation de la classe décrivant le graphique d'un objet
musical
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 08/10/96
// Modifications:
// Commentaires:
#ifdef MUSIQUE

#include "xvt.h"

#ifdef TEST_MUSIQUE
#include "TesMuDef.h"
#include CBrouillonXVT_i
#else
#include "XvtDef.h"
#include CFenetreXVT_i
#endif

#include "MusiDef.h"
#include CGraphiqueMusical_i
#include CCachePixmap_i
#include CPartition_i
#include Musique_i

// Fonction privée d'affichage d'une ligne vue comme un polygone
void CGraphiqueMusical::AffichePolygone
      (PNT *pointDepart, PNT *pointArrivee, CPEN *stylo, WINDOW fenetre)
      const
{     PNT polygone[5],              // Tableau des points du polygone
          point2,                   // Points intermédiaires du polygone
          point3;
      CBRUSH brosse;                // Brosse employée pour remplir le
                                       polygone
      short epaisseur = stylo->     // Epaisseur du trait du stylo
            width;
      COLOR couleur = stylo->color; // Couleur d'affichage
      stylo->width = 1;             // Initialisation du stylo (pour le
                                       dessin du polygone
      point2.h = pointDepart->h;    // On calcule les points intermédiaires
      point2.v = pointDepart->v + epaisseur - 1;
      point3.h = pointArrivee->h;
```

```
      point3.v = pointArrivee->v + epaisseur - 1;
      *polygone = *pointDepart;     // On calcule les coordonnées du
                                       polygone
      *(polygone + 1) = *pointArrivee;
      *(polygone + 2) = point3;
      *(polygone + 3) = point2;
      *(polygone + 4) = *pointDepart;
      brosse.pat = PAT_SOLID;       // On initialise les paramètres de
                                       dessin pour effacer
      brosse.color = COLOR_BLACK;
      stylo->color = COLOR_BLACK;
      xvt_dwin_set_cbrush(fenetre, &brosse);
      xvt_dwin_set_cpen(fenetre, stylo);
      xvt_dwin_set_draw_mode(fenetre, M_CLEAR);
      xvt_dwin_draw_polygon         // On efface la zone du polygone
            (fenetre, polygone, 5);
      brosse.color = stylo->color = // On change les paramètres de dessin
            couleur;
      xvt_dwin_set_cbrush(fenetre, &brosse);
      xvt_dwin_set_cpen(fenetre, stylo);
      xvt_dwin_set_draw_mode(fenetre, M_OR);
      xvt_dwin_draw_polygon         // On dessine le polygone en couleur
            (fenetre, polygone, 5);
}

// Constructeur
CGraphiqueMusical::CGraphiqueMusical(long abscisse, long ordonnee)
      : CObjet()
{     positionX = abscisse;         // Initialisation des membres de la
      positionY = ordonnee;         // classe
}

// Fonction protégée
// On renvoie le point supérieur gauche de l'objet relatif à la fenetre
void CGraphiqueMusical::Position (long decalageGauche, long decalageHaut,
PNT &positionFenetre) const
{     positionFenetre.h = (short) (positionX + decalageGauche);
      positionFenetre.v = (short) (positionY + decalageHaut);
}

// Fonction protégée de dessin d'une image dans la fenêtre à l'emplacement
// des coordonnées du graphique musical + un certain décalage, en le
// redimensionnant à l'echelle de la zone musicale
void CGraphiqueMusical::AfficheImage
      (int numeroRessource, int largeur, int hauteur,
       short decalageX, short decalageY,
       long decalageGauche, long decalageHaut,
       WINDOW fenetre, const CPartition *partition) const
{     PNT positionFenetre;          // Position relative dans la fenêtre
      float echelle;                // Echelle de redimensionnement des
                                       graphiques musicaux
      COLOR couleur;                // Couleur d'affichage du dessin
      RCT rectangleCible,           // Emplacement du résultat
          rectangleSource;          // Emplacement dans la pixmap de dessin
                                       intermédiaire
      XVT_PIXMAP pixmap;            // Pixmap de dessin intermédiaire
      CCachePixmap *cache =         // On accède à la pixmap en noir et
                                       blanc
            partition->CachePixmap();
      Position                      // On calcule la position du graphique
                                       relative à la fenêtre visible
            (decalageGauche, decalageHaut, positionFenetre);
      echelle = partition->         // On calcule l'échelle
            Echelle();
```

```
      xvt_rect_set(&rectangleSource,// On initialise les dimensions du
                                    rectangle source
            0, 0, (short)(largeur * echelle), (short)(hauteur * echelle));
      xvt_rect_set(&rectangleCible, // On calcule les dimensions du
                                    rectangle cible
            positionFenetre.h + decalageX,
            positionFenetre.v + decalageY,
            positionFenetre.h + decalageX + short(largeur * echelle),
            positionFenetre.v + decalageY + short(hauteur * echelle));
      if (numeroRessource == DESSIN_CERCLE)
            couleur = COLOR_RED;    // Si le dessin est le cercle de panneau
                                    d'attention, la pixmap est rouge,
      else couleur = partition->    // Sinon, on trouve la couleur de la
                                    pixmap dans la zone
            OutilsDessin()->fore_color;
      if ((pixmap = cache->Pixmap   // On accède à la pixmap en noir et
                                    blanc
            (numeroRessource, couleur, echelle, largeur, hauteur, FALSE))
            == NULL_PIXMAP) return;
      xvt_dwin_set_draw_mode        // On se prépare à nettoyer
                                    l'emplacement du dessin
            (fenetre, M_CLEAR);
      xvt_dwin_draw_pmap            // On efface les points du dessin aux
                                    dimensions voulues dans la fenêtre
            (fenetre, pixmap, &rectangleCible, &rectangleSource);
      if ((pixmap = cache->Pixmap   // On accède à la pixmap en couleur
            (numeroRessource, couleur, echelle, largeur, hauteur, TRUE)) ==
            NULL_PIXMAP) return;
      xvt_dwin_set_draw_mode        // On se prépare à afficher le dessin
            (fenetre, M_OR);
      xvt_dwin_draw_pmap            // On affiche la pixmap couleur dans la
                                    fenêtre
            (fenetre, pixmap, &rectangleCible, &rectangleSource);
}

// Fonction protégée d'affichage d'une ligne dans l'objet
void CGraphiqueMusical::AfficheLigne
      (const PNT *pointDebut, const PNT *pointFin, CPEN *stylo,
       long decalageGauche, long decalageHaut,
       WINDOW fenetre, const CPartition *partition) const
{     PNT positionFenetre,          // Position relative dans la fenêtre
          pointDepart,              // Point de départ et d'arrivée
                                    effectifs de la ligne
          pointArrivee;
      CPEN styloNoir = *stylo;
      styloNoir.color = COLOR_BLACK;
      Position                      // On calcule la position du graphique
                                    relative à la fenêtre visible
            (decalageGauche, decalageHaut, positionFenetre);
      pointDepart.h =               // On calcule les points de départ
            positionFenetre.h + pointDebut->h;
      pointDepart.v =
            positionFenetre.v + pointDebut->v;
      pointArrivee.h =              // On calcule les points d'arrivée
            positionFenetre.h + pointFin->h;
      pointArrivee.v =
            positionFenetre.v + pointFin->v;
      if (stylo->style == P_SOLID)  // Si le trait est plein,
      {     AffichePolygone         // on affiche la ligne en affichant un
                                    polygone
                  (&pointDepart, &pointArrivee, stylo, fenetre);
            return;
      }
      xvt_dwin_set_cpen(fenetre,    // On initialise le stylo de la fenêtre
```

```
            &styloNoir);
      xvt_dwin_set_draw_mode(fenetre, M_CLEAR);
      xvt_dwin_draw_set_pos         // On positionne le stylo au point de
                                       départ
            (fenetre, pointDepart);
      xvt_dwin_draw_line            // On trace la ligne
            (fenetre, pointArrivee);
      xvt_dwin_set_cpen(fenetre,    // On initialise le stylo de la fenêtre
            stylo);
      xvt_dwin_set_draw_mode(fenetre, M_OR);
      xvt_dwin_draw_set_pos (fenetre, pointDepart);
      xvt_dwin_draw_line(fenetre, pointArrivee);
}

// Fonction protégée d'affichage d'un nombre dans un objet
void CGraphiqueMusical::AfficheNombre
      (short nombre, const XVT_FNTID police,
       short translationX, short translationY,
       long decalageGauche, long decalageHaut,
       WINDOW fenetre, const CPartition *partition) const
{     float echelle = partition->   // Echelle de redimensionnement des
                                       graphiques musicaux
            Echelle();
      PNT positionFenetre,          // Position relative dans la fenêtre
          point;                    // Point d'affichage du nombre
      const DRAW_CTOOLS *outils     // Outils de l'environnement de la
                                       partition
            = partition->OutilsDessin();
      DRAW_CTOOLS outilsUtilises;   // Outils utilisés pour l'affichage du
                                       texte
      outilsUtilises.pen = outils->pen;
      outilsUtilises.brush = outils->brush;
      outilsUtilises.mode = outils->mode;
      outilsUtilises.fore_color = outils->fore_color;
      outilsUtilises.back_color = outils->back_color;
      outilsUtilises.opaque_text = FALSE;
      char buffer[8];               // Buffer de conversion du nombre en
                                       ascii
      Position                      // On calcule la position du graphique
                                       relative à la fenêtre visible
            (decalageGauche, decalageHaut, positionFenetre);
      xvt_dwin_set_font(fenetre,    // On initialise la police de caractères
                                       de la fenêtre
            police);
      point.h = positionFenetre.h + // On calcule les points de départ par
                                       rapport au point de début (mise à
                                       l'échelle)
              short(translationX * echelle);
      point.v = positionFenetre.v +
              (int) xvt_dwin_get_font_size_mapped(fenetre) +
              short(translationY * echelle);
      sprintf(buffer, "%d", nombre);// On convertit le nombre en ascii
      xvt_dwin_set_draw_ctools      // On initialise les outils de dessin
            (fenetre, (DRAW_CTOOLS *) &outilsUtilises);
      xvt_dwin_draw_text            // On affiche le texte
            (fenetre, point.h, point.v, buffer, -1);
}

#endif
```

Annexe 3 : CGraphiqueLiaison et CGraphiqueNOlet

Cette annexe comporte un extrait du fichier CGraAutr.h et CGraAutr.cpp

CGraAutr.h

```
// Nature: Définition des classes décrivant les objets graphiques
// barre de mesure, liaison, n-olet et barre de notes.
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 16/10/96
// Modifications:
// Commentaires:

#ifndef CGraphiqueAutres_H
#define CGraphiqueAutres_H

#include "MusiDef.h"
#include CGraphiqueMusical_i

class CZoneMusiqueXVT;
[...]
class CGraphiqueLiaison : public CGraphiqueMusical
{     private:                        // Implémentation en mémoire:
            RCT ovale;                //    Ovale qui contient l'arc de
                                      liaison
            PNT pointDebut;           //    Points de début et de fin de la
                                      liaison
            PNT pointFin;

      public:                         // Interface standard:
            CGraphiqueLiaison         //    Constructeur
                  (long, long, RCT *, PNT *, PNT *);
            void Affiche              //    Affichage de la liaison
                  (long, long, const CPartition *, WINDOW) const;
};

class CGraphiqueNolet : public CGraphiqueMusical
{     private:                        // Implémentation en mémoire:
            PNT pointDebut;           //    Points de début et de fin de la
                                      ligne longitudinale
            PNT pointFin;
            BOOLEAN orientationBasse;
                                      //    Détermine si l'orientation des
                                      barres verticales du n-olet
            short n;                  //    Définit le type de n-olet

      public:                         // Interface standard:
            CGraphiqueNolet           //    Constructeur
                  (long, long, PNT *, PNT *, short, BOOLEAN);
            void Affiche              //    Affichage du n-olet
                  (long, long, const CPartition *, WINDOW) const;
};
[...]
#endif
```

CGraAutr.cpp

```
// Nature: Implémentation des classes décrivant les graphiques de type
// barre de mesure, liaison, n-olet, barre reliant un groupe de notes
```

```
// et du symbole continuateur de mesure.
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 16/10/96
// Modifications:
// Commentaires:
#ifdef MUSIQUE

#include "xvt.h"
#include "musique.h"

#include "MusiDef.h"
#include CGraphiqueAutres_i
#include CPartition_i
#include ConstantesMusique_i
[...]
// Cas de la liaison
// Constructeur
CGraphiqueLiaison::CGraphiqueLiaison
      (long positionX, long positionY, RCT *ovaleDesire,
       PNT *pointDebutDesire, PNT *pointFinDesire)
      : CGraphiqueMusical(positionX, positionY)
{     ovale = *ovaleDesire;
      pointDebut = *pointDebutDesire;
      pointFin = *pointFinDesire;
}

// Affichage de la liaison
void CGraphiqueLiaison::Affiche
      (long decalageGauche, long decalageHaut,
       const CPartition *partition, WINDOW fenetre) const
{     PNT positionFenetre;          // Position relative dans la fenêtre
      RCT rectangleOvaleCible;      // Rectangle où se dessinera l'ovale
      int debutX, debutY,           // Coordonnées des points de début et de
                                       fin effectifs de la liaison
          finX, finY;
      Position                      // On calcule la position du graphique
                                       relative à la fenêtre visible
            (decalageGauche, decalageHaut, positionFenetre);
      CPEN stylo;                   // Stylo avec lequel on va dessiner la
                                       liaison
      stylo.style = P_SOLID;        // On initialise le stylo
      stylo.color = partition->OutilsDessin()->fore_color;
      stylo.width = 1;
      stylo.pat = PAT_SOLID;
      if (ovale.left == 0 &&        // Si on a un ovale vide (dépassement de
                                       capacité lors du découpage)
            ovale.top == 0 && ovale.right == 0 && ovale.bottom == 0)
      {     AfficheLigne            // On affiche une ligne du point de
                                       début au point de fin de liaison
                  (&pointDebut, &pointFin, &stylo, decalageGauche,
                   decalageHaut, fenetre, partition);
            return;
      }
      xvt_dwin_set_cpen(fenetre,    // On place le stylo de la fenêtre
            &stylo);
      debutX =                      // On calcule les points de départ
             positionFenetre.h - ovale.left + pointDebut.h;
      debutY = positionFenetre.v - ovale.top + pointDebut.v;
      finX = positionFenetre.h      // On calcule les points d'arrivée
            - ovale.left + pointFin.h;
      finY =      positionFenetre.v - ovale.top + pointFin.v;
      rectangleOvaleCible.top =     // On calcule les coordonnées du
                                       rectangle contenant l'ovale cible
            positionFenetre.v;
```

```
      rectangleOvaleCible.left =
            positionFenetre.h;
      rectangleOvaleCible.bottom =
            positionFenetre.v - ovale.top + ovale.bottom;
      rectangleOvaleCible.right =
            positionFenetre.h - ovale.left + ovale.right;
      xvt_dwin_set_draw_mode(fenetre, M_COPY);
      xvt_dwin_draw_arc             // On trace la liaison
            (fenetre, &rectangleOvaleCible, debutX, debutY, finX, finY);
}

// Cas du n-olet
// Constructeur
CGraphiqueNolet::CGraphiqueNolet
      (long positionX, long positionY,
       PNT *pointDebutDesire, PNT *pointFinDesire,
       short nDesire, BOOLEAN orientationBasseDesiree)
      : CGraphiqueMusical(positionX, positionY)
{     pointDebut =                  // Initialisation des différents
                                       paramètres
            *pointDebutDesire;
      pointFin = *pointFinDesire;
      n = nDesire;
      orientationBasse = orientationBasseDesiree;
}

// Affichage du n-olet
// Attention, lorsqu'on parle de coéfficient angulaire, le point d'origine
// des axes se trouve en haut à gauche de l'objet
void CGraphiqueNolet::Affiche
      (long decalageGauche, long decalageHaut,
       const CPartition *partition, WINDOW fenetre) const
{     PNT point;                    // Point de début ou de fin de ligne
      CPEN stylo;                   // Stylo avec lequel on va dessiner les
                                       lignes
      int hauteurMoyenne = (pointDebut.v + pointFin.v) / 2;
      int largeurMoyenne = (pointDebut.h + pointFin.h) / 2;
      float coefficientAngulaire =  // Coéfficient angulaire de la barre
                                       diagonale
            float(pointDebut.v - pointFin.v) /
            float(pointDebut.h - pointFin.h);
      float echelle = partition->   // Echelle de redimensionnement des
                                       graphiques musicaux
            Echelle();
      XVT_FNTID police =            // On initialise la petite police
            xvt_font_create();
      xvt_font_set_family(police, XVT_FFN_TIMES);
      xvt_font_set_size(police, long(echelle * 48) / 3);
      xvt_font_set_style(police, XVT_FS_NONE);
#if (XVTWS == WINWS)
      stylo.style = P_DOT;          // On initialise le stylo pour WINDOWS
#elif (XVTWS == MACWS)
      stylo.style = P_SOLID;        // On initialise le stylo pour MAC
#elif (XVTWS == NTWS)
      stylo.style = P_DOT;          // On initialise le stylo pour UNIX
#endif
      stylo.color = partition->OutilsDessin()->fore_color;
      stylo.width = 1;
      stylo.pat = PAT_SOLID;
      point.h = pointDebut.h;       // On initialise le point pour tracer la
                                       ligne verticale gauche
      point.v = orientationBasse ? pointDebut.v + short(20 * echelle)
                           : pointDebut.v - short(20 * echelle);
      AfficheLigne                  // On trace la ligne verticale gauche
```

```
            (&pointDebut, &point, &stylo, decalageGauche,
             decalageHaut, fenetre, partition);
          point.h = pointFin.h;       // On initialise le point pour tracer la
                                         ligne verticale droite
      point.v = orientationBasse ? pointFin.v + short(20 * echelle)
                           : pointFin.v - short(20 * echelle);
      AfficheLigne                    // On trace la ligne verticale droite
            (&pointFin, &point, &stylo, decalageGauche,
             decalageHaut, fenetre, partition);
      point.h =                       // On initialise le point pour tracer la
                                         ligne diagonale gauche
            largeurMoyenne - short(12 * echelle);
      point.v = pointDebut.v + short((point.h - pointDebut.h) *
            coefficientAngulaire);
      AfficheLigne                    // On trace la ligne diagonale gauche
            (&pointDebut, &point, &stylo, decalageGauche,
             decalageHaut, fenetre, partition);
      point.h =                       // On initialise le point pour tracer la
                                         ligne diagonale droite
            largeurMoyenne + short(12 * echelle);
      point.v = pointFin.v + short((point.h - pointFin.h) *
coefficientAngulaire);
      AfficheLigne                    // On trace la ligne diagonale droite
            (&pointFin, &point, &stylo, decalageGauche,
             decalageHaut, fenetre, partition);
      AfficheNombre                   // On affiche le n du n-olet
            (n, police,
             short(largeurMoyenne / echelle) - 6,
             short(hauteurMoyenne / echelle) - 6,
             decalageGauche, decalageHaut,
             fenetre, partition);
      xvt_font_destroy(police);
}
[...]
#endif
```

Annexe 4 : CGraphiqueNote

Cette annexe contient le listing des fichiers CGraNote.h, CGraNote.cpp, CGraPoin.h et CGraPoin.cpp.

CGraNote.h

```
// Nature: Définition de la classe décrivant l'objets graphiques notes
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 17/10/96
// Modifications:
// Commentaires:

#ifndef CGraphiqueNotes_H
#define CGraphiqueNotes_H

#include "MusiDef.h"
#include CGraphiquePointe_i

class CGraphiquePortee;
class CZoneMusiqueXVT;

class CGraphiqueNote : public CGraphiquePointe
{     protected:                    // Implémentation en mémoire:
            short nombreLignes;     //    Nombre de lignes de portée
                                          supplémentaires,
                                    //    si négatif, sous la portée, sinon
                                          au-dessus de la portée
            int duree;              //    Durée de la note en 128ème de
                                          noire
            const CGraphiquePortee *//    Portée dans laquelle la note est
                                          incluse
                  saPortee;

                                    // Fonctions à usage privé
            BOOLEAN AfficheLignes   //    Fonction d'affichage de lignes
                  (short, long, long,
                   WINDOW, const CPartition *) const;

      public:                       // Interface standard:
            CGraphiqueNote          //    Constructeur
                  (long, long, short, int, short);
            virtual void Affiche    //    Affichage de la note
                  (long, long, const CPartition *, WINDOW) const = 0;
            virtual BOOLEAN Note()  //    L'objet est-il un objet graphique
                                          de note ?
                  const {return TRUE;}
            virtual BOOLEAN         //    L'objet est-il un objet graphique
                                          de note avec hampe ?
                  NoteHampe() const {return FALSE;}
            virtual int             //    Décalage proportionnel de la note
                                          lors du chewingum
                  DecalageIdeal() const {return duree;}
            inline int Duree ()     //    Accès à la durée de la note en
                                          128ième de noire
                  const {return duree;}
            virtual void            //    Transformation des moustaches
                                          logiques en moustaches graphiques
                  AjouteMoustachesGraphiques(float){}
            virtual BOOLEAN         //    La note a-t-elle une hampe haute ?
                  HampeHaute() {return FALSE;}
```

```
            virtual short           //    Accès au nombre de moustaches
                                          logiques
                  NombreMoustaches() const {return 0;}
            virtual long            //    Ordonnée du point le plus haut de
                                          la note
                  OrdonneeMinimale(float) const {return positionY;}
            virtual long            //    Ordonnée du point le plus bas de
                                          la note
                  OrdonneeMaximale(float) const = 0;
            inline void SaPortee    //    Changement de la portée incluant
                                          la note
                  (CGraphiquePortee *portee) {saPortee = portee;}
};

class CGraphiqueNoteHampe : public CGraphiqueNote
{     protected:                    // Implémentation en mémoire:
            short hauteurHampe;     //    hauteur de la hampe (pixels) de
                                          l'extrémité de la hampe à la base
                                          de la note
            BOOLEAN hampeHaute;     //    Si la direction de la hampe est le
                                          haut

                                    // Fonctions à usage privé
            void AfficheHampe       //    Fonction d'affichage de la hampe
                  (short, short, short, long, long,
                   WINDOW, const CPartition *) const;

      public:                       // Interface standard:
            CGraphiqueNoteHampe     //    Constructeur
                  (long, long, short, BOOLEAN, short, int, short);
            virtual BOOLEAN         //    L'objet est-il un objet graphique
                                          de note avec hampe ?
                  NoteHampe()const {return TRUE;}
            inline BOOLEAN          //    Accès au drapeau de position de la
                                          hampe
                  HampeHaute() const {return hampeHaute;}
            inline void HampeHaute  //    Changement du drapeau de position
                                          de la hampe
                  (BOOLEAN position) {hampeHaute = position;}
            inline short            //    Accès à la hauteur de la hampe
                  HauteurHampe() const {return hauteurHampe;}
            inline void HauteurHampe//    Changement de la hauteur de la
                                          hampe
                  (short hauteur) {hauteurHampe = hauteur;}
            virtual long            //    Ordonnée du point le plus haut de
                                          la note
                  OrdonneeMinimale(float) const;
            virtual long            //    Ordonnée du point le plus bas de
                                          la note
                  OrdonneeMaximale(float) const;
};

class CGraphiqueNoteMoustachue : public CGraphiqueNoteHampe
{     protected:                    // Implémentation en mémoire:
            short                   //    Nombre de moustaches affichées de
                                          la note
                  nombreMoustachesGraphiques,
                  nombreMoustachesLogiques;
                                    //    Nombre de moustaches "logiques" de
                                          la note (en rapport au rythme)


                                    // Fonctions à usage privé
            void AfficheMoustaches  //    Fonction d'affichage des
```

```
                                        moustaches
                  (long, long, WINDOW, const CPartition *) const;

      public:                       // Interface standard:
            CGraphiqueNoteMoustachue// Constructeur
                  (long, long, short, BOOLEAN, short, short, int, short);

            virtual void            //    Transformation des moustaches
                                          logiques en moustaches graphiques
                  AjouteMoustachesGraphiques(float);
            inline virtual short    //    Accès au nombre de moustaches
                                          logiques
                  NombreMoustaches() const
                  {return nombreMoustachesLogiques;}
};

class CGraphiqueNoire : public CGraphiqueNoteMoustachue
{     public:                       // Interface standard:
            CGraphiqueNoire         //    Constructeur
                  (long, long, short, BOOLEAN, short, short, int, short);
            virtual void Affiche    //    Affichage de la note
                  (long, long, const CPartition *, WINDOW) const;
};

class CGraphiqueBlanche : public CGraphiqueNoteHampe
{     public:                       // Interface standard:
            CGraphiqueBlanche       //    Constructeur
                  (long, long, short, BOOLEAN, short, int, short);
            virtual void Affiche    //    Affichage de la note
                  (long, long, const CPartition *, WINDOW) const;
            virtual long            //    Ordonnée du point le plus haut de
                                          la note
                  OrdonneeMinimale(float) const;
            virtual long            //    Ordonnée du point le plus bas de
                                          la note
                  OrdonneeMaximale(float) const;
};

class CGraphiqueRonde : public CGraphiqueNote
{     public:                       // Interface standard:
            CGraphiqueRonde         //    Constructeur
                  (long, long, short, int, short);
            virtual void Affiche    //    Affichage de la note
                  (long, long, const CPartition *, WINDOW) const;
            virtual long            //    Ordonnée du point le plus bas de
                                          la note
                  OrdonneeMaximale(float) const;
};

class CGraphiqueCarree : public CGraphiqueNote
{     public:                       // Interface standard:
            CGraphiqueCarree        //    Constructeur
                  (long, long, short, int, short);
            virtual void Affiche    //    Affichage de la note
                  (long, long, const CPartition *, WINDOW) const;
            virtual long            //    Ordonnée du point le plus bas de
                                          la note
                  OrdonneeMaximale(float) const;
};

#endif
```

CGraNote.cpp

```
// Nature: Implémentation de la classe décrivant le graphique d'une note
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 17/10/96
// Modifications:
// Commentaires:
#ifdef MUSIQUE

#include "xvt.h"
#include "musique.h"

#include "MusiDef.h"
#include CGraphiqueNotes_i
#include CGraphiquePortee_i
#include CPartition_i
#include ConstantesMusique_i

// Cas de la note
// Fonction protégée d'affichage des lignes supplémentaires de portée
BOOLEAN CGraphiqueNote::AfficheLignes
      (short largeur, long decalageGauche, long decalageHaut,
       WINDOW fenetre, const CPartition *partition) const
{     float echelle = partition->   // Echelle de redimensionnement des
                                      graphiques musicaux
            Echelle();
      short nombreLignesSupplementaires,
                                    // Compteur de lignes affichées
            nombreLignesAbsolu,     // Nombre de lignes à afficher
            abscisse,               // Abscisse du point gauche des lignes
            ordonneeLigne0;         // Ordonnée de la ligne de la portée la
                                      plus proche
      PNT pointDebut,               // Point de début et de fin des lignes
          pointFin;
      CPEN stylo;                   // Stylo utilisé pour afficher les
                                      lignes
      if (!saPortee) return FALSE;  // Si la portée incluant la note n'est
                                      pas initialisée, on rapporte l'erreur
      abscisse =                    // On calcule l'abscisse du point gauche
                                      des lignes
            -short((((COEFFICIENT_LARGEUR_LIGNE_NOTE - 1) * largeur) / 2) *
            echelle);               // Ce coefficient est le rapport entre
                                      la largeur de la ligne et la largeur
                                      du corps de la note
      ordonneeLigne0 =              // On calcule l'ordonnée de la ligne de
                                      la portée la plus proche
            nombreLignes > 0?
            short(saPortee->PositionY() - positionY):
            short(saPortee->PositionY() + short((HAUTEUR_INTERLIGNE << 2) *
            echelle) - positionY);
      pointDebut.h = abscisse;      // On initialise les abscisses des
                                      points
      pointFin.h = short((largeur + ((COEFFICIENT_LARGEUR_LIGNE_NOTE - 1) *
            largeur) / 2) * echelle);
      stylo.style = P_SOLID;        // On initialise le stylo
      stylo.pat = PAT_SOLID;
      stylo.color = partition->OutilsDessin()->fore_color;
      stylo.width = 1;
      nombreLignesSupplementaires   // On initialise le compteur de lignes
                                      supplémentaires
            = 0;
      nombreLignesAbsolu =          // On calcule le nombre de lignes absolu
            nombreLignes > 0? nombreLignes: -nombreLignes;
      while (++nombreLignesSupplementaires <= nombreLignesAbsolu)
      {     pointDebut.v =          // On initialise les ordonnées des
```

```
                                        points de début et de fin de ligne
                  pointFin.v =
                  (nombreLignes > 0)?
                  ordonneeLigne0 - short((nombreLignesSupplementaires *
                  HAUTEUR_INTERLIGNE) * echelle):
                  ordonneeLigne0 + short((nombreLignesSupplementaires *
                  HAUTEUR_INTERLIGNE) * echelle);
            AfficheLigne            // On affiche la ligne
                  (&pointDebut, &pointFin, &stylo,
                   decalageGauche, decalageHaut,
                   fenetre, partition);
      }
      return TRUE;
}

// Constructeur
CGraphiqueNote::CGraphiqueNote
      (long abscisse, long ordonnee, short nombreLignesDesirees,
       int dureeDesiree, short nombrePoints)
      : CGraphiquePointe(abscisse, ordonnee, nombrePoints)
{     nombreLignes =                // On initialise le graphique demandé
            nombreLignesDesirees;
      saPortee = NULL;
      duree = dureeDesiree;
}

// Cas de la note avec hampe
// Fonction privée d'affichage de la hampe
void CGraphiqueNoteHampe::AfficheHampe
      (short arrondiNote, short hauteurNote, short largeurNote,
       long decalageGauche, long decalageHaut,
       WINDOW fenetre, const CPartition *partition) const
{     PNT pointDebut,               // Points de début et de fin de ligne
          pointFin;
      float echelle = partition->   // Echelle de redimensionnement des
                                       graphiques musicaux
            Echelle();
      CPEN stylo;                   // Stylo avec lequel on va dessiner la
                                       hampe
      pointDebut.v =                // On initialise les points de début et
                                       de fin de la hampe
            hampeHaute? short(hauteurNote * echelle) - arrondiNote:
            arrondiNote;
      pointDebut.h =
            hampeHaute? short(largeurNote * echelle) - 1: 0;
      pointFin.v =
            hampeHaute? short(hauteurNote * echelle) - hauteurHampe:
            hauteurHampe;
      pointFin.h = pointDebut.h;
      stylo.style = P_SOLID;        // On initialise le stylo
      stylo.pat = PAT_SOLID;
      stylo.color = partition->OutilsDessin()->fore_color;
      stylo.width = 1;
      AfficheLigne                  // On trace la hampe
            (&pointDebut, &pointFin, &stylo,
             decalageGauche, decalageHaut,
             fenetre, partition);
}

// Constructeur
CGraphiqueNoteHampe::CGraphiqueNoteHampe
      (long positionX, long positionY,
       short hauteurDesiree, BOOLEAN positionDesiree,
       short nombreLignes, int rythme, short nombrePoints)
```

```
      : CGraphiqueNote(positionX, positionY, nombreLignes, rythme,
nombrePoints)
{     hauteurHampe = hauteurDesiree;
      hampeHaute = positionDesiree;
}

// Fonction virtuelle : Renvoie l'ordonnée du point supérieur de la note
long CGraphiqueNoteHampe::OrdonneeMinimale(float echelle) const
{     return hampeHaute?
            positionY + short(HAUTEUR_NOIRE * echelle) - hauteurHampe:
            positionY;
}

// Fonction virtuelle : Renvoie l'ordonnée du point inférieur de la note
long CGraphiqueNoteHampe::OrdonneeMaximale(float echelle) const
{     return hampeHaute?
            positionY + short(HAUTEUR_NOIRE * echelle):
            positionY + hauteurHampe;
}

// Cas de la note avec moustaches
// Fonction privée d'affichage des moustaches
void CGraphiqueNoteMoustachue::AfficheMoustaches
      (long decalageGauche, long decalageHaut,
       WINDOW fenetre, const CPartition *partition) const
{     int dessin;                   // Numero de la ressource de la
                                       moustache à afficher
      float echelle = partition->   // Echelle de redimensionnement des
                                       graphiques musicaux
            Echelle();
      short largeur,                // Dimensions du dessin
            hauteur,
            numeroMoustache,        // Numéro de la moustache courante
            ecartement = short      // On calcule l'écartement des
                                       moustaches
                  (ESPACE_ENTRE_MOUSTACHES * echelle);
      short abscisse,               // Coordonnées des moustaches par
                                       rapport aux coordonnées de l'objet
            ordonnee;
      abscisse =                    // On calcule les coordonnees de
                                       l'extrémité de la hampe pour y
                                       ajouter les moustaches
            hampeHaute? short(LARGEUR_NOIRE * echelle) - 1: 0;
      ordonnee =                    // 26 est la hauteur de la moustache
                                       pour les notes avec hampe en bas
            hampeHaute? -hauteurHampe + short(HAUTEUR_NOIRE * echelle):
            hauteurHampe - short(26 * echelle);
      if (hampeHaute)               // On initialise les différentes
                                       variables
      {     dessin = DESSIN_MOUSTACHE_HAUT;
            largeur = LARGEUR_MOUSTACHE_HAUT;
            hauteur = HAUTEUR_MOUSTACHE_HAUT;
      }
      else
      {     dessin = DESSIN_MOUSTACHE_BAS;
            largeur = LARGEUR_MOUSTACHE_BAS;
            hauteur = HAUTEUR_MOUSTACHE_BAS;
            ecartement = -ecartement;
      }
      for (numeroMoustache =        // Boucle d'affichage des premières
                                       moustaches
            nombreMoustachesGraphiques;
           numeroMoustache > 1;
           numeroMoustache--)
```

```
        {     AfficheImage(           // On affiche une moustache dans la
                                         fenêtre
                    dessin, largeur, hauteur,
                    abscisse, ordonnee,
                    decalageGauche, decalageHaut,
                    fenetre, partition);
              ordonnee += ecartement; // On change la hauteur de la moustache
                                         suivante
        }
        if (numeroMoustache)
        {     AfficheImage(           // On affiche la derniere moustache dans
                                         la fenêtre
                    hampeHaute? DESSIN_DERNIERE_MOUSTACHE_HAUT:
                                DESSIN_DERNIERE_MOUSTACHE_BAS,
                    hampeHaute? LARGEUR_DERNIERE_MOUSTACHE_HAUT:
                                LARGEUR_DERNIERE_MOUSTACHE_BAS,
                    hampeHaute? HAUTEUR_DERNIERE_MOUSTACHE_HAUT:
                                HAUTEUR_DERNIERE_MOUSTACHE_BAS,
                    abscisse,
                    hampeHaute?       // (deuxième cas : -8 car la dernière
                                         moustache bas a une plus grande
                                         hauteur que les autres moustaches
                    ordonnee: ordonnee - short(8 * echelle),
                    decalageGauche,   decalageHaut, fenetre, partition);
        }
}

// Constructeur
CGraphiqueNoteMoustachue::CGraphiqueNoteMoustachue
        (long positionX, long positionY,
         short hauteurHampe, BOOLEAN hampeHaute,
         short nombreMoustachesDesirees,
         short nombreLignes, int rythme, short nombrePoints)
        : CGraphiqueNoteHampe(positionX, positionY,
              hauteurHampe, hampeHaute, nombreLignes, rythme, nombrePoints)
{     nombreMoustachesLogiques = nombreMoustachesDesirees;
      nombreMoustachesGraphiques = 0;
}

// Transformation des moustaches logiques en moustaches graphiques
void CGraphiqueNoteMoustachue::AjouteMoustachesGraphiques(float echelle)
{     nombreMoustachesGraphiques =  // On effectue la transformation
              nombreMoustachesLogiques;
      hauteurHampe +=               // On agrandit la hampe
                                       proportionnellement au nombre des
                                       moustaches graphiques
              short((nombreMoustachesGraphiques * (ESPACE_ENTRE_MOUSTACHES))
              * echelle);
}

// Cas de la note noire
// Constructeur
CGraphiqueNoire::CGraphiqueNoire
        (long positionX, long positionY,
         short hauteurHampeDesiree, BOOLEAN hampeHauteDesiree,
         short nombreMoustachesDesirees,
         short nombreLignesDesirees, int rythme, short nombrePoints)
        : CGraphiqueNoteMoustachue(positionX, positionY,
              hauteurHampeDesiree, hampeHauteDesiree,
              nombreMoustachesDesirees, nombreLignesDesirees, rythme,
              nombrePoints)
{
}
```

```
// Affichage d'une note noire
void CGraphiqueNoire::Affiche
      (long decalageGauche, long decalageHaut,
       const CPartition *partition,
       WINDOW fenetre) const
{     float echelle = partition->   // Echelle de redimensionnement des
                                    graphiques musicaux
            Echelle();
      short arrondiNote =           // Distance entre le bord de la note et
                                    le point de rattachement à la hampe
            short(HAUTEUR_NOIRE * echelle / 2.0);
      if (nombreLignes)             // Si on a des lignes de portée
                                    supplémentaires,
            AfficheLignes           // on affiche les lignes de portee
                                    supplémentaires
                  (LARGEUR_NOIRE, decalageGauche, decalageHaut, fenetre,
                  partition);
      AfficheImage(                 // On affiche le dessin de la noire dans
                                    la fenêtre
            DESSIN_NOIRE,
            LARGEUR_NOIRE, HAUTEUR_NOIRE, 0, 0,
            decalageGauche, decalageHaut, fenetre, partition);
      AfficheHampe(arrondiNote, HAUTEUR_NOIRE, LARGEUR_NOIRE,
            decalageGauche, decalageHaut, fenetre, partition);
      AfficheMoustaches             // On affiche les moustaches
            (decalageGauche, decalageHaut, fenetre, partition);
      AffichePoints(LARGEUR_NOIRE,  // On affiche les points de la notes
            HAUTEUR_NOIRE >> 1, decalageGauche, decalageHaut, fenetre,
            partition);
}

// Cas de la note blanche
// Constructeur
CGraphiqueBlanche::CGraphiqueBlanche
      (long positionX, long positionY,
       short hauteurHampeDesiree, BOOLEAN hampeHauteDesiree,
       short nombreLignesDesirees, int rythme, short nombrePoints)
      : CGraphiqueNoteHampe(positionX, positionY,
            hauteurHampeDesiree, hampeHauteDesiree, nombreLignesDesirees,
            rythme, nombrePoints)
{
}

// Affichage d'une note blanche
void CGraphiqueBlanche::Affiche
      (long decalageGauche, long decalageHaut,
       const CPartition *partition,
       WINDOW fenetre) const
{     float echelle = partition->   // Echelle de redimensionnement des
                                    graphiques musicaux
            Echelle();
      short arrondiNote =           // Distance entre le bord de la note et
                                    le point de rattachement à la hampe
            short(HAUTEUR_BLANCHE * echelle / 2.0);
      if (nombreLignes)             // Si on a des lignes de portée
                                    supplémentaires,
           AfficheLignes            // on affiche les lignes de portee
                                    supplémentaires
                  (LARGEUR_BLANCHE, decalageGauche, decalageHaut, fenetre,
                  partition);
     AfficheImage(                  // On affiche le dessin de la blanche
                                    dans la fenêtre
            DESSIN_BLANCHE,
            LARGEUR_BLANCHE, HAUTEUR_BLANCHE, 0, 0,
```

```
            decalageGauche,   decalageHaut, fenetre, partition);
      AfficheHampe(arrondiNote, HAUTEUR_BLANCHE, LARGEUR_BLANCHE,
            decalageGauche, decalageHaut, fenetre, partition);
      AffichePoints(LARGEUR_BLANCHE,// On affiche les points de la notes
            HAUTEUR_BLANCHE >> 1, decalageGauche, decalageHaut, fenetre,
            partition);
}

// Fonction virtuelle : Renvoie l'ordonnée du point supérieur de la note
long CGraphiqueBlanche::OrdonneeMinimale(float echelle) const
{     return hampeHaute?
            positionY + short(HAUTEUR_BLANCHE * echelle) - hauteurHampe:
            positionY;
}

// Fonction virtuelle : Renvoie l'ordonnée du point inférieur de la note
long CGraphiqueBlanche::OrdonneeMaximale(float echelle) const
{     return hampeHaute?
            positionY + short(HAUTEUR_BLANCHE * echelle):
            positionY + hauteurHampe;
}

// Cas de la ronde
// Constructeur
CGraphiqueRonde::CGraphiqueRonde
      (long positionX, long positionY,
       short nombreLignesDesirees, int rythme, short nombrePoints)
      : CGraphiqueNote(positionX, positionY, nombreLignesDesirees, rythme,
nombrePoints)
{
}

// Affichage d'une ronde
void CGraphiqueRonde::Affiche
      (long decalageGauche, long decalageHaut,
       const CPartition *partition,
       WINDOW fenetre) const
{     if (nombreLignes)            // Si on a des lignes de portée
                                      supplémentaires,
            AfficheLignes          // on affiche les lignes de portee
                                      supplémentaires
                  (LARGEUR_RONDE, decalageGauche, decalageHaut, fenetre,
                  partition);
      AfficheImage(                // On affiche le dessin de la ronde dans
                                      la fenêtre
            DESSIN_RONDE,
            LARGEUR_RONDE, HAUTEUR_RONDE,
            0, 0, decalageGauche, decalageHaut, fenetre, partition);
      AffichePoints(LARGEUR_RONDE,  // On affiche les points de la notes
            HAUTEUR_RONDE >> 1, decalageGauche, decalageHaut, fenetre,
            partition);
}

// Fonction virtuelle : Renvoie l'ordonnée du point inférieur de la note
long CGraphiqueRonde::OrdonneeMaximale(float echelle) const
{     return      positionY + short(HAUTEUR_RONDE * echelle);
}

// Cas de la carrée
// Constructeur
CGraphiqueCarree::CGraphiqueCarree
      (long positionX, long positionY,
       short nombreLignesDesirees, int rythme, short nombrePoints)
      : CGraphiqueNote(positionX, positionY, nombreLignesDesirees, rythme,
```

```
            nombrePoints)
{
}

// Affichage d'une carrée
void CGraphiqueCarree::Affiche
      (long decalageGauche, long decalageHaut,
       const CPartition *partition,
       WINDOW fenetre) const
{     if (nombreLignes)            // Si on a des lignes de portée
                                      supplémentaires,
            AfficheLignes          // on affiche les lignes de portee
                                      supplémentaires
                  (LARGEUR_CARREE, decalageGauche, decalageHaut, fenetre,
                   partition);
      AfficheImage(                // On affiche le dessin de la carrée
                                      dans la fenêtre
            DESSIN_CARREE,
            LARGEUR_CARREE, HAUTEUR_CARREE,
            0, 0, decalageGauche, decalageHaut, fenetre, partition);
      AffichePoints(LARGEUR_CARREE, // On affiche les points de la notes
            HAUTEUR_CARREE >> 1, decalageGauche, decalageHaut, fenetre,
            partition);
}

// Fonction virtuelle : Renvoie l'ordonnée du point inférieur de la note
long CGraphiqueCarree::OrdonneeMaximale(float echelle) const
{     return      positionY + short(HAUTEUR_CARREE * echelle);
}

#endif
```

CGraPoin.h

```
// Nature: Définition de la classe décrivant l'objets graphiques pointés
// (notes ou silences)
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 20/01/97
// Modifications:
// Commentaires:

#include CGraphiqueMusical_i
#include "Musique.h"

#ifndef CGraphiquePointe_H
#define CGraphiquePointe_H

class CGraphiquePointe : public CGraphiqueMusical
{     protected:                   // Implémentation en mémoire:
            short nombrePoints;    //    Nombre de points

                                   // Fonctions à usage privé
            void AffichePoints     //    Fonction d'affichage des points
                  (short, short, long, long,
                   WINDOW, const CPartition *) const;

      public:                      // Interface standard:
            CGraphiquePointe       //    Constructeur
                  (long, long, short);
            virtual void Affiche   //    Affichage de la note
                  (long, long, const CPartition *, WINDOW) const = 0;
};
#endif
```

CGraPoin.cpp

```
// Nature: Implémentation de la classe décrivant l'objets graphiques
pointés
// (notes ou silences)
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 20/01/97
// Modifications:
// Commentaires:

#include "MusiDef.h"
#include CGraphiquePointe_i
#include ConstantesMusique_i
#include CPartition_i

// Fonction privée d'affichage de points
void CGraphiquePointe::AffichePoints
      (short decalageHorizontal, short milieuGraphique,
       long decalageGauche, long decalageHaut,
       WINDOW fenetre, const CPartition *partition) const
{     float echelle = partition->   // Echelle de redimensionnement des
                                      graphiques musicaux
            Echelle();
      for (short numeroPoint = 0;
           numeroPoint < nombrePoints;)
      {     AfficheImage(DESSIN_POINT, LARGEUR_POINT, HAUTEUR_POINT,
                  short(echelle * (decalageHorizontal + ++numeroPoint *
                  ESPACE_AVANT_POINT - LARGEUR_POINT)),
                  short(echelle * (milieuGraphique - 2)),
                  decalageGauche, decalageHaut,
                  fenetre, partition);
      }
}

// Constructeur
CGraphiquePointe::CGraphiquePointe
      (long positionX, long positionY, short nombrePointsDesire)
      : CGraphiqueMusical(positionX, positionY)
{     nombrePoints = nombrePointsDesire;
}
```

Annexe 5 : CDecoupageMusical

Cette annexe contient le listing des fichiers CDecMusi.h, CDecMusi.cpp, CDecPart.h, CDecPart.cpp, CDecPort.h, CDecPort.cpp, CDecMesu.h et CDecMesu.cpp.

CDecMusi.h

```
// Nature: Définition de la classe décrivant un objet musical de découpage
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 25/10/96
// Modifications:
// Commentaires:

#ifndef CDecoupageMusical_H
#define CDecoupageMusical_H

#include "xvt.h"

#include "WaveDef.h"
#include CObjet_i

class CDecoupageMusical : public CObjet
{     protected:                    // Implémentation en mémoire:
            long positionX,         //    Coordonnee de l'objet dans la
                                          fenêtre
                 positionY;
            int largeur;            //    Largeur de l'objet de découpage

                                    // Fonctions à usage privé
      public:                       // Interface standard:
            CDecoupageMusical       //    Constructeur
                  (long, long, int);
            inline int Largeur()    //    Accès à la largeur de l'objet de
                                          découpage
                  const {return largeur;}
};

#endif
```

CDecMusi.cpp

```
// Nature: Implémentation de la classe décrivant une objet de découpage
// musical
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 25/10/96
// Modifications:
// Commentaires:
#ifdef MUSIQUE

#include "MusiDef.h"
#include CDecoupageMusical_i

// Constructeur
CDecoupageMusical::CDecoupageMusical
      (long abscisse, long ordonnee, int largeurDesiree)
      : CObjet()
{     positionX = abscisse;
      positionY = ordonnee;
      largeur = largeurDesiree;
}
#endif
```

CDecPart.h

```
// Nature: Définition de la classe décrivant une partition de découpage
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 24/10/96
// Modifications:
// Commentaires:

#ifndef CPortitionDecoupage_H
#define CPortitionDecoupage_H

#include "xvt.h"

#ifdef TEST_MUSIQUE
#include "TesMuDef.h"
#else
#include "PresDef.h"
#endif
#include CZoneMusiqueXVT_i

#include "MusiDef.h"
#include CPartition_i
#include CDecoupageMusical_i
#include CMusique_i

#include "WaveDef.h"
#include CVecteur_i

class CDecoupagePartition : public CDecoupageMusical
{     private:                        // Implémentation en mémoire:
            CPartition *partition;    //    Partition associée
            short ton;                //    Ton courant
            CMusiqueMesure mesure;    //    Mesure courante
            CMusiqueCle cle;          //    Clé courante
            int tempo;                //    Tempo courant
            int largeurDisponible;    //    Largeur disponible à la partition
            int hauteurPartition;
            CVecteur                  //    Liste temporaire de liaisons
                  brouillonLiaisons;
            CVecteur portees;         //    Liste des portées de la partition

                                      // Fonctions privées:
            BOOLEAN                   //    Ajouts des éventuels
                                            avertissements
                  AjouteAvertissements(const unsigned char *&, int&);
            BOOLEAN RemplisPortees    //    Remplissage des portées à partir
                                            de la mélodie
                  (const unsigned char *&);
            int EcartePortees(int);   //    Positionne verticalement les
                                            portées
            void Justifie();          //    Justification des graphiques pour
                                            qu'ils occupent toute la largeur
                                            de la partition
            BOOLEAN                   //    Ajout des graphiques réels des
                                            barres de notes
                  AjouteBarresNotes();
            BOOLEAN AjouteNOlets();   //    Ajout des graphiques des n-olets
            BOOLEAN                   //    Ajout des graphiques réels des
                                            liaisons
                  AjouteLiaisons();

      public:                         // Interface standard:
            CDecoupagePartition       //    Constructeur
                  (CPartition *, int, long, long);
            ~CDecoupagePartition(); //    Destructeur
```

```
            BOOLEAN Ajoute          //    Ajoute une mélodie à la partition
                  (const unsigned char *&, int);
            BOOLEAN Cle             //    Change la clé courante
                  (nomCle, short);
            inline CPartition *     //    Accès à la partition associée
                  Partition() const {return partition;}
            inline void Ton         //    Modification du ton courant
                  (short tonalite){ton = tonalite;}
            inline short Ton()      //    Accès au ton courant
                  const {return ton;}
            inline CMusiqueCle *    //    Accès à la clé courante
                  Cle() {return &cle;}
            inline CMusiqueMesure * //    Accès à la mesure courante
                  Mesure() {return &mesure;}
            inline float Echelle()  //    Accès à l'échelle de
                                         redimensionnement des graphiques
                                         musicaux
                  const {return partition->Echelle();}
            inline CVecteur *       //    Accès au vecteur où sont stockés
                                         les objets graphiques musicaux
                  ObjetsGraphiques() {return partition->Partition();}
            inline int              //    Accès largeur disponible à la
                                         partition
                  LargeurDisponible() const {return largeurDisponible;}
            inline CVecteur *       //    Accès à la liste temporaire de
                                         liaisons
                  BrouillonLiaisons() {return &brouillonLiaisons;}
            inline int              //    Accès à la hauteur de la partition
                  HauteurPartition() const {return hauteurPartition;}
};
#endif
```

CDecPart.cpp

```
// Nature: Implémentation de la classe décrivant une partition de découpage
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 24/10/96
// Modifications:
// Commentaires:
#ifdef MUSIQUE

#include "WaveDef.h"
#include CVecteurIterateur_i

#include "MusiDef.h"
#include CDecoupagePartition_i
#include CDecoupagePortee_i
#include CGraphiqueTempo_i
#include CGraphiqueAttention_i
#include CGraphiqueAutres_i
#include CBrouillonGraphiqueHorizontal_i
#include ConstantesMusique_i

// Fonction privée d'ajout des graphiques d'avertissement dans la partition
BOOLEAN CDecoupagePartition::AjouteAvertissements
      (const unsigned char *&melodie, int &largeurIndicationsDiverses)
{     CGraphiqueMusical *graphique; // Graphique musical inséré dans la
                                      partition
      float echelle = partition->   // Echelle de redimensionnement des
                                      graphiques musicaux
            Echelle();
      if ((*melodie & 0x02u)==      // On n'a pas d'armature
            0x00u)
      {     if (!(graphique = new   // On crée le graphique du cercle rouge
```

```
                  CGraphiqueCercleRouge(largeurIndicationsDiverses, 0L)))
                  return FALSE;
            partition->Partition()  // On le note dans la partition
                  ->push(graphique);
            if (!(graphique = new   // On crée le graphique de l'icone
                                    d'absence d'armatures
                  CGraphiqueIconeArmatures
                        (largeurIndicationsDiverses, 0L))) return FALSE;
            partition->Partition()->// On le note dans la partition
                  push(graphique);
            largeurIndicationsDiverses += int((LARGEUR_CERCLE + 10) *
                  echelle);
      }
      if ((*melodie & 0x01u)== 0x00u)// On n'a pas de rythme
      {     if (!(graphique = new   // On crée le graphique du cercle rouge
                  CGraphiqueCercleRouge(largeurIndicationsDiverses, 0L)))
                  return FALSE;
            partition->Partition()->// On le note dans la partition
                  push(graphique);
            if (!(graphique = new   // On crée le graphique de l'icone
                                    d'absence de rythmes
                  CGraphiqueIconeRythmes(largeurIndicationsDiverses, 0L)))
                        return FALSE;
            partition->Partition()->// On le note dans la partition
                  push(graphique);
            largeurIndicationsDiverses +=
                  int((LARGEUR_CERCLE + 10) * echelle);
      }
      if ((*melodie++ & 0x04u)      // Les indications de mesures ne sont
                                    pas contraignantes
            == 0x00u)
      {     if (!(graphique = new   // On crée le graphique du cercle rouge
                  CGraphiqueCercleRouge(largeurIndicationsDiverses, 0L)))
                  return FALSE;
            partition->Partition()->// On le note dans la partition
                  push(graphique);
            if (!(graphique = new   // On crée le graphique de l'icone de
                                    mesures non contraignantes
                  CGraphiqueIconeMesure(largeurIndicationsDiverses, 0L)))
                  return FALSE;
            partition->Partition()  // On le note dans la partition
                  ->push(graphique);
            largeurIndicationsDiverses +=
                  int((LARGEUR_CERCLE + 10) * echelle);
      }
      return TRUE;
}

// Fonction privée de décodage de la mélodie pour créer
// tous les objets graphiques non horizontaux
// Renvoie TRUE si tout s'est bien passé
BOOLEAN CDecoupagePartition::RemplisPortees(const unsigned char *&melodie)
{     CDecoupagePortee *portee;     // Portée courante dans la partition
      const unsigned char *         // Premier code dans une portée
            codeDebutPortee = melodie;
      float echelle = partition->   // Echelle de redimensionnement des
                                    graphiques musicaux
            Echelle();
      long dureeTick;               // Durée d'un tick en microsecondes
      CGraphiqueMusical *graphique; // Graphique musical inséré dans la
                                    partition
      int largeurIndicationsDiverses// Largeur des indications de tempo,
                                    présence de rythmes et d'armatures
                                    // que l'on initialise pour commencer à
```

```
                                    gauche de la clé
            = int(LARGEUR_CLE_UT * echelle);
      if ((*codeDebutPortee & 0xF0u)// On vérifie que le premier code est
                                    un code d'entête
            != 0xC0u) return FALSE;
      if (!AjouteAvertissements     // On vérifie la présence d'armatures et
                                    de rythmes.  Au cas échéant, on
                                    affiche des avertissements
            (codeDebutPortee, largeurIndicationsDiverses)) return FALSE;
      dureeTick =                   // On calcule le tempo
            *codeDebutPortee++ << 8;
      dureeTick |=                  // Formule: tempo = 60 000 000 / (96 *
                                    durée d'un tick en µsec)
            *codeDebutPortee++;
      tempo = dureeTick? int(625000L / dureeTick): 0;
      if (tempo)                    // Si on a une indication du tempo
      {     if ((                   // On arrête si on dépasse le bord
                                    gauche
                                    // Remarque : 50 est la largeur
                                    appoximative de l'indication de tempo
                  largeurIndicationsDiverses + int(50 * echelle))
                  > largeurDisponible) return FALSE;
            if (!(graphique = new   // On crée le graphique de tempo
                  CGraphiqueTempo(tempo, largeurIndicationsDiverses, 0L)))
                  return FALSE;
            partition->Partition()->// On le note dans la partition
                  push(graphique);
      }
      while ((*codeDebutPortee & 0xF0u) != 0xF0u)
      {     if (!(portee =          // On crée une nouvelle portée
                  new CDecoupagePortee(this, 0L, 0L))) return FALSE;
            if (!portee->Ajoute     // On ajoute la mélodie commençant par
                                    ce code à cette portée
                  (codeDebutPortee))
            {     delete portee;
                  return FALSE;
            }
            portees.push(portee);   // On note la nouvelle portée
      }
      melodie = codeDebutPortee;    // On avance dans la mélodie
      return TRUE;
}

// Fonction privée de placement vertical des portées.
// Renvoie la hauteur totale de la partition
int CDecoupagePartition::EcartePortees(int decalage)
{     float echelle = Echelle();    // Echelle de redimensionnement des
                                    graphiques musicaux
      int hauteur                   // Hauteur de la portée
            = int((HAUTEUR_CERCLE + 4) * echelle);
      CVecteurIterateur             // Itérateur sur les portées de la
                                    partition
            prochainePortee(portees);
      CDecoupagePortee *portee;     // Portée à positionner verticalement
      while (portee =               // En itérant sur les portées de la
                                    partition,
            (CDecoupagePortee *) prochainePortee())
      {     portee->                // On met à jour les dépassements de la
                                    portée
                  ChangeDepassements();
            hauteur += int(echelle *// On met à jour la hauteur de la
                                    portée suivante
                portee->DepassementHaut());
            portee->                // On translate l'ordonnée des objets
```

```
                TranslationVerticale(decalage + hauteur);
            hauteur +=              // On met à jour la hauteur de la portée
                                       suivante
                int(echelle * ((HAUTEUR_INTERLIGNE << 2) +
                portee->DepassementBas())) + ESPACE_ENTRE_PORTEES;
        }
        return hauteur - ESPACE_ENTRE_PORTEES;
}

// Fonction privée de repositionnement des objets graphiques pour qu'ils
// occupent toute la largeur de la partition
void CDecoupagePartition::Justifie()
{       CVecteurIterateur           // Itérateur sur les portées de la
                                       partition
            prochainePortee(portees);
        CDecoupagePortee *portee;   // Portée à justifier
        while (portee =             // En itérant sur les portées de la
                                       partition,
            (CDecoupagePortee *) prochainePortee())
                portee->            // on traite cette portée
                    Justifie();
}

// Fonction privée d'ajout des graphiques des barres de notes
BOOLEAN CDecoupagePartition::AjouteBarresNotes()
{       CVecteurIterateur           // Itérateur sur les portées de la
                                       partition
            prochainePortee(portees);
        CDecoupagePortee *portee;   // Portée à laquelle on veut ajouter les
                                       barres de notes
        while (portee =             // En itérant sur les portées de la
                                       partition,
            (CDecoupagePortee *) prochainePortee())
            if (!portee->           // on traite cette portée
                AjouteBarresNotes()) return FALSE;
        return TRUE;
}

// Fonction privée d'ajout des graphiques des n-olet
BOOLEAN CDecoupagePartition::AjouteNOlets()
{       CVecteurIterateur           // Itérateur sur les portées de la
                                       partition
            prochainePortee(portees);
        CDecoupagePortee *portee;   // Portée à laquelle on veut ajouter les
                                       n-olet
        while (portee =             // En itérant sur les portées de la
                                       partition,
            (CDecoupagePortee *) prochainePortee())
            if (!portee->           // on traite cette portée
                AjouteNOlets()) return FALSE;
        return TRUE;
}

// Fonction privée d'ajout des graphiques réels des liaisons
BOOLEAN CDecoupagePartition::AjouteLiaisons()
{       CVecteurIterateur           // Itérateur sur les brouillons de
                                       liaison de la partition
            liaisonSuivante(brouillonLiaisons);
        CBrouillonGraphiqueLiaison  // Liaison à traiter
            *liaison;
        BOOLEAN liaisonTraitee;
        CVecteurIterateur           // Itérateur sur les portées de la
                                       partition
            porteeSuivante(portees);
```

```
      CDecoupagePortee *portee;     // Portée courante
      while (liaison =              // Tant qu'il reste des liaisons
            (CBrouillonGraphiqueLiaison *) liaisonSuivante())
      {     liaisonTraitee = FALSE;
            porteeSuivante.reset();
            while ((portee = (CDecoupagePortee *) porteeSuivante())
                  && !liaisonTraitee)
            {     if (!portee->     // On essaie d'ajouter cette liaison à
                                    la portée courante
                        TraiteLiaison(partition->Partition(), liaison,
                              liaisonTraitee))
                              return FALSE;
            }
            if (!liaisonTraitee)    // Si on n'a réussi à la traiter sur
                                    aucune portée, on a une erreur
                  return FALSE;
      }
      return TRUE;
}

// Constructeur
CDecoupagePartition::CDecoupagePartition
      (CPartition *partitionDesiree, int largeurOctroyee, long abscisse,
       long ordonnee)
      : CDecoupageMusical(abscisse, ordonnee, largeurOctroyee),
        portees(8),
        cle(cleSol, 2),
        mesure(4, 4, FALSE),
        brouillonLiaisons()
{     partition = partitionDesiree;
      largeurDisponible = largeurOctroyee;
      ton = 0;
}

// Destructeur
CDecoupagePartition::~CDecoupagePartition()
{     portees.clearAndDestroy();
      brouillonLiaisons.clearAndDestroy();
}

// Ajout de graphiques musicaux dans la partition
BOOLEAN CDecoupagePartition::Ajoute
      (const unsigned char *&melodie, int decalage)
{     CGraphiqueFond *fondMelodie;  // Graphique de fond de la mélodie
      CVecteurIterateur             // Itérateur sur les portées de la
                                    partition
            porteeSuivante(portees);
      if (!RemplisPortees(melodie)) // On crée tous les graphiques sauf les
                                    barres de note et les liaisons
            return FALSE;
      Justifie();                   // On justifie la mélodie pour qu'elle
                                    prenne toute la place dans la portée
      if (!AjouteBarresNotes())     // On ajoute les graphiques réels des
                                    barres de notes
            return FALSE;
      if (!AjouteNOlets())          // On ajoute les graphiques des n-olets
            return FALSE;
      if (!AjouteLiaisons())        // On ajoute les graphiques réels des
                                    liaisons
            return FALSE;
      hauteurPartition =            // On positionne verticalement les
                                    portées et on initialise la hauteur
                                    de la zone
            EcartePortees(decalage);
```

```
      if (!(fondMelodie =           // On crée un fond pour cette mélodie
            new CGraphiqueFond(positionX, positionY, largeurDisponible + 1,
                  hauteurPartition)))
                  return FALSE;
      partition->Partition()->      // On le note en début de partition
            insertAt(0, fondMelodie);
      return TRUE;
}

// Changement de la clé courante
BOOLEAN CDecoupagePartition::Cle(nomCle nom, short ligne)
{     cle.Nom(nom);
      cle.Ligne(ligne);
      return TRUE;
}

#endif
```

CDecPort.h

```
// Nature: Définition de la classe décrivant une portée de découpage
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 21/10/96
// Modifications:
// Commentaires:

#ifndef CDecoupagePortee_H
#define CDecoupagePortee_H

#include "xvt.h"

#include "MusiDef.h"
#include CDecoupageMusical_i
#include CDecoupagePartition_i
#include CBrouillonGraphiqueHorizontal_i

#include "WaveDef.h"
#include CVecteur_i

class CGraphiquePortee;

class CDecoupagePortee : public CDecoupageMusical
{     private:                      // Implémentation en mémoire:
            CDecoupagePartition *   //    Partition associée
                  partition;
            short depassementHaut,  //    Hauteur des objets graphiques
                                    //    dépassant au-dessus  de la portée
                  depassementBas;   //                en-dessous
            CGraphiquePortee *      //    Référence de la portée graphique
                  saPortee;
            CVecteur mesures;       //    Liste des mesures de la portée

                                    // Fonctions à usage privé
            BOOLEAN AjouteLiaison   //    Ajout d'une liaison dont les
                                    //    indices sont connus
                  (CBrouillonGraphiqueLiaison *, int, int);
            BOOLEAN                 //    L'indice correspond, dans le
                                    //    vecteur des graphiques de la
                                    //    partition, à un graphique de la
                                    //    portée
                  IndiceDansPortee(int) const;

      public:                       // Interface standard:
            CDecoupagePortee        //    Constructeur
```

```
            (CDecoupagePartition *, long, long);
        ~CDecoupagePortee();    //    Destructeur
        void ChangeDepassements //    Mise à jour des dépassements
            ();
        BOOLEAN Ajoute          //    Ajoute une mélodie à la portée
            (const unsigned char *&);
        void                    //    Translate tous les ordonnées des
                                      objets graphiques de la portée
            TranslationVerticale(long);
        void Justifie();        //    Justification des objets
                                      graphiques pour qu'ils occupent
                                      toute la largeur de la portée
        BOOLEAN                 //    Ajout des graphiques de barres de
                                      notes
            AjouteBarresNotes();
        BOOLEAN AjouteNOlets(); //    Ajout des graphiques des n_olets
        BOOLEAN TraiteLiaison   //    Ajout d'une liaison dans la portée
            (CVecteur *, CBrouillonGraphiqueLiaison *, BOOLEAN &);
        inline float Echelle()  //    Accès à l'échelle de
                                      redimentionnement des graphiques
                                      de la partition
            const {return partition->Echelle();}
        inline CGraphiquePortee //    Accès à la portée graphique
            * SaPortee() const {return saPortee;}
        inline int              //    Calcul de la largeur disponible
                                      dans la portée
            LargeurDisponible() const
            {return partition->LargeurDisponible() - largeur;}
        inline short            //    Accès au décalage haut
            DepassementHaut() const {return depassementHaut;}
        inline short            //    Accès au décalage bas
            DepassementBas() const {return depassementBas;}
        inline int              //    Accès au nombre de mesures dans le
                                      portée
            NombreMesures(){return mesures.entries();}
        inline CDecoupageMesure* PremiereMesure() const
            {return (CDecoupageMesure*) mesures.first();}
        inline CDecoupageMesure* DerniereMesure() const
            {return (CDecoupageMesure *) mesures.last();}
);
#endif
```

CDecPort.cpp

```
// Nature: Implémentation de la classe décrivant une portée de découpage
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 21/10/96
// Modifications:
// Commentaires:
#ifdef MUSIQUE

#include "MusiDef.h"
#include CDecoupagePortee_i
#include CGraphiqueNotes_i
#include CGraphiquePortee_i
#include CGraphiqueAutres_i
#include CDecoupageMesure_i
#include ConstantesMusique_i

#include "WaveDef.h"
#include CVecteurIterateur_i

// Fonction privée d'ajout d'une liaison dont on connait les
// indices de début et de fin
```

```
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupagePortee::AjouteLiaison
      (CBrouillonGraphiqueLiaison* liaison, int indiceDebut, int indiceFin)
{     if (!liaison->MiseAuPropre    // On note la liaison dans le vecteur
                                    des objets graphiques
            (this, indiceDebut, indiceFin)) return FALSE;
      if (depassementHaut <         // On change éventuellement les
                                    dépassements de la portée
            liaison->DepassementHaut())
                  depassementHaut = liaison->DepassementHaut();
      if (depassementBas < liaison->DepassementBas())
            depassementBas = liaison->DepassementBas();
      return TRUE;
}

// Fonction privée d'indication si un graphique du vecteur des
// graphiques de la partition correspond à un graphique de la portée
// Renvoie TRUE si cela correspond.
BOOLEAN CDecoupagePortee::IndiceDansPortee(int indice) const
{     if (indice < (                // L'indice n'est pas dans la portée si
                                    il est plus petit que le premier
                                    graphique de la première mesure de la
                                    portée
            (CDecoupageMesure *)mesures.first())->IndicePremiereNote())
            return FALSE;
      if (indice > (                // L'indice n'est pas dans la portée si
                                    il est plus grand que le dernier
                                    graphique de la dernière mesure de la
                                    portée
            (CDecoupageMesure *)mesures.last())->IndiceDerniereNote())
            return FALSE;
      return TRUE;                  // Sinon, l'indice est dans la portée
}

// Constructeur
CDecoupagePortee::CDecoupagePortee
      (CDecoupagePartition *partitionUtilisee, long abscisse,
       long ordonnee)
      : CDecoupageMusical(abscisse, ordonnee, 0), mesures(8)
{     partition = partitionUtilisee;// On note la partition associée
      depassementHaut =             // On commence avec une portée vierge
            depassementBas = 0;
      saPortee = NULL;              // On n'a pas encore de portee graphique
}

// Destructeur
CDecoupagePortee::~CDecoupagePortee()
{     mesures.clearAndDestroy();
}

// Mise à jour des dépassements
void CDecoupagePortee::ChangeDepassements()
{     CVecteurIterateur             // Itérateur sur le vecteur des mesures
            mesureSuivante(mesures);
      CDecoupageMesure *mesure;     // Mesure traitée
      while (mesure = (CDecoupageMesure *)
            mesureSuivante())
      {     if (depassementHaut <   // On met à jour le dépassement haut si
                                    le nouveau est plus grand que
                                    l'ancien
                  mesure->DepassementHaut())
                  depassementHaut = mesure->DepassementHaut();
            if (depassementBas <    // On met à jour le dépassement bas si
                                    le nouveau est plus grand que
```

```
                                    l'ancien
                mesure->DepassementBas())
                depassementBas = mesure->DepassementBas();
        }
}

// Fonction d'ajout d'une melodie.
// Le pointeur melodie référence le premier code de la portée
// Renvoie TRUE si tout s'est bien passé
BOOLEAN CDecoupagePortee::Ajoute(const unsigned char *&melodie)
{       int largeurDisponible           // Largeur restant disponible dans la
                                           portée
            = LargeurDisponible();
        CDecoupageMesure *mesure;       // Mesure courante dans la portée
        const unsigned char *           // Premier code dans une mesure
            codeDebutMesure = melodie;
        BOOLEAN debutPortee = TRUE,     // drapeau: on est en début de portée
            mesureComplete;             // drapeau: la mesure se termine par un
                                           barre
        if (!(saPortee = new            // On crée la portée graphique
            CGraphiquePortee (LargeurDisponible(), 0L, 0L))) return FALSE;
        partition->                     // On la note dans la liste des objets
                                           graphiques
            ObjetsGraphiques()->push(saPortee);
        while ((*codeDebutMesure & 0xF0u) != 0xF0u)
        {       if (!(mesure =          // On crée une nouvelle mesure
                    new CDecoupageMesure(partition, this, largeur, 0L)))
                        return FALSE;
                if (debutPortee)        // Si on est en début de portée
                {       if (!mesure->   // On insère l'armature
                            AjouteArmature(codeDebutMesure))
                        {       delete mesure;
                                return FALSE;
                        }
                        debutPortee =   // On n'est plus en début de portée
                            FALSE;
                }
                if (!mesure->Ajoute     // On ajoute à cette mesure une partie
                                           de la mélodie
                    (codeDebutMesure, mesureComplete)) return FALSE;
                if (!mesureComplete) break;
                mesures.push(mesure);   // On la note
                largeur += mesure->     // On met à jour la largeur de la portée
                    Largeur();
                melodie = codeDebutMesure;
        }
        if ((*codeDebutMesure & 0xF0u)// Si on est arrivés à la dernière
                                           mesure et qu'elle est complète,
            == 0xF0u && mesureComplete) return TRUE;
        if ((*codeDebutMesure & 0xF0u)// Si on est arrivés à la dernière
                                           mesure
            == 0xF0u)
        {       mesures.push(mesure);   // On note la mesure
                largeur +=              // On met à jour la largeur de la portée
                    mesure->Largeur();
                melodie =               // On avance dans la mélodie
                    codeDebutMesure;
                return TRUE;
        }
        if (mesures.entries())          // Si on n'a pas eu assez de place dans
                                           la première mesure
        {       mesure->                // On supprime les objets graphiques de
                                           cette mesure
                    EffaceObjetsGraphiques();
```

```
            delete mesure;          // On supprime la mesure sans avancer
                                       dans la mélodie
            return TRUE;            // Et on a fini
      }
      if (codeDebutMesure ==        // Si on n'a traité aucun code sur cette
                                       portée, echec
            melodie)
      {     delete mesure;
            return FALSE;
      }
      if (!mesure->                 // On ajoute un continuateur de mesure
            AjouteContinuateur())
      {     delete mesure;
            return FALSE;
      }                             // On met à jour s'il y a lieu les
                                       dépassements
      mesures.push(mesure);         // On note la mesure
      largeur += mesure->Largeur(); // On met à jour la largeur de la portée
      melodie = codeDebutMesure;    // On avance dans la mélodie
      return TRUE;
}

// Fonction de translation verticale de tous les objets graphiques
// de la portée
void CDecoupagePortee::TranslationVerticale(long decalage)
{     CVecteurIterateur             // Itérateur sur les mesures de la
                                       portée
            prochaineMesure(mesures);
      CDecoupageMesure *mesure;     // Mesure à translater
      saPortee->PositionY           // On translate l'objet graphique de la
                                       portée
            (saPortee->PositionY() + decalage);
      while (mesure =               // En itérant sur les mesures de la
                                       portée,
            (CDecoupageMesure *)prochaineMesure())
            mesure->                // on translate l'ordonnée des objets
                  TranslationVerticale(decalage);
}

// Repositionnement des objets graphiques pour qu'ils occupent toute la
largeur de la portée
void CDecoupagePortee::Justifie()
{     CVecteurIterateur             // Itérateur sur les mesures de la
                                       portée
            prochaineMesure(mesures);
      CDecoupageMesure *mesure;     // Mesure à ranger
      short supplement,             // Pixels supplémentaires accordé à la
                                       mesure
            numeroMesure = 0,       // Numéro d'ordre dans le vecteur de la
                                       mesure traitée
            decalage = 0;           // Décalage subi par les dernières
                                       mesures suite à l'accroissement de
                                       taille des premières
      while (mesure =               // En itérant sur les mesures de la
                                       portée,
            (CDecoupageMesure *)prochaineMesure())
      {     numeroMesure++;         // On est sur la mesure suivante dans le
                                       vecteur
            supplement =            // On calcule le supplément de pixel que
                                       l'on va donner à la mesure
                  (LargeurDisponible() + 1) /
                  (mesures.entries() - numeroMesure + 1);
            mesure->                // On réorganise les graphiques de cette
                                       mesure en fonction des pixels
```

```
                                    accordés et du décalage subi
                ArrangeGraphiques(decalage, supplement);
            decalage += supplement; // On met à jour le décalage pour les
                                    mesures suivantes
            largeur += supplement;  // On met à jour la largeur de la portée
        }
}

// Ajout des graphiques des barres de notes
BOOLEAN CDecoupagePortee::AjouteBarresNotes()
{     CVecteurIterateur             // Itérateur sur les mesures de la
                                    portée
            prochaineMesure(mesures);
      CDecoupageMesure *mesure;     // Mesure à ranger
      while (mesure =               // En itérant sur les mesures de la
                                    portée,
            (CDecoupageMesure *)prochaineMesure())
            if (!mesure->           // On traite cette mesure
                  AjouteBarresNotes()) return FALSE;
      return TRUE;
}

// Ajout des graphiques des n-olet
BOOLEAN CDecoupagePortee::AjouteNOlets()
{     CVecteurIterateur             // Itérateur sur les mesures de la
                                    portée
            prochaineMesure(mesures);
      CDecoupageMesure *mesure;     // Mesure à ranger
      while (mesure =               // En itérant sur les mesures de la
                                    portée,
            (CDecoupageMesure *)prochaineMesure())
            if (!mesure->           // On traite cette mesure
                  AjouteNOlets()) return FALSE;
      return TRUE;
}

// Ajout du graphique d'une liaison
// Renvoie TRUE si tout s'est bien passé
BOOLEAN CDecoupagePortee::TraiteLiaison
      (CVecteur *graphiques, CBrouillonGraphiqueLiaison *liaison, BOOLEAN
&liaisonTraitee)
{     int indiceDebut,              // Indice de début et de fin de la
                                    liaison
          indiceFin;
      if (!(liaison->               // Si l'indice de la première note est
                                    plus grand que l'indice de la
                                    première note de la portée,
            IndicePremiereNote(graphiques) <
            ((CDecoupageMesure *) mesures.first())->IndicePremiereNote()))
      {     if (!IndiceDansPortee(  // Si cette indice n'est pas dans la
                                    portée, on a terminé
                  liaison->IndicePremiereNote(graphiques)))
                        return TRUE;
            indiceDebut = liaison-> // Sinon, le début de la liaison
                                    commence sur cet indice
                  IndicePremiereNote(graphiques);
      }                             // Si la liaison a déjà été commencée
                                    sur une portée précédente,
      else indiceDebut = -1;        // la liaison commence avant la première
                                    note de cette portée
      indiceFin = liaison->         // On initialise l'indice de fin à
                                    l'indice de la dernière note de la
                                    liaison
            IndiceDerniereNote(graphiques);
```

```
      if (IndiceDansPortee          // Si la liaison se termine dans cette
                                       portée,
            (indiceFin))
      {     liaisonTraitee = TRUE;  // La liaison a été traitée totalement
            return AjouteLiaison    // On ajoute la liaison
                  (liaison, indiceDebut, indiceFin);
      }
      return AjouteLiaison          // Sinon, on ajoute une liaison qui se
                                       termine après la dernière note de la
                                       portée
            (liaison, indiceDebut, -1);
}
#endif
```

CDecMesu.h

```
// Nature: Définition de la classe décrivant une mesure de découpage
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 24/10/96
// Modifications:
// Commentaires:

#ifndef CMesureDecoupage_H
#define CMesureDecoupage_H

#include "xvt.h"

#include "MusiDef.h"
#include CGraphiqueMusical_i
#include CDecoupageMusical_i

#include "WaveDef.h"
#include CVecteur_i

class CGraphiqueNote;
class CZoneMusiqueXVT;
class CDecoupagePartition;
class CDecoupagePortee;
class CMusiqueNote;
class CVecteurIterateurGraphiques;

class CDecoupageMesure : public CDecoupageMusical
{     private:                      // Implémentation en mémoire:
            CDecoupagePartition *   //    Partition dans laquelle s'insère
                                          la mesure
                  partition;
            CDecoupagePortee *      //    Portée dans laquelle s'insère la
                                          mesure
                  portee;
            short depassementHaut,  //    Nombre de pixels qui dépassent en
                                          haut et en bas de la mesure
                  depassementBas,
                  nombreNotesNOlet; //    Nombre de notes dans le n-olet
                                          courant (= 0 si on ne se trouve
                                          pas dans un n-olet)
            int positionDansPortee; //    Début de la mesure dans la portée
            short                   //    Altération des notes de la mesure
                                          à un moment donné
                  alterationsCourantes[7];
            CVecteur                //    Liste temporaire de n-olets
                  brouillonNOlets;
            CVecteur                //    Liste temporaire des barres de
                                          notes
                  brouillonBarresNotes;
```

```
        CVecteur graphiques;    //    Liste des objets graphiques de la
                                      mesure

                                // Fonctions à usage privé:
        void ChangeDepassements //    Mise à jour les dépassements si un
                                      des paramètres est plus grand que
                                      les dépassements courants
              (short, short);
        BOOLEAN AjouteGraphique //    Ajout d'un graphique dans la
                                      mesure
              (CGraphiqueMusical *, short, short, short);
        BOOLEAN AjouteTonalite  //    Ajout de la tonalité du morceau
              ();
        BOOLEAN AjouteSilence   //    Ajout d'un silence dans la mesure
              (int, int);
        BOOLEAN                 //    Ajout de bécarres après la
                                      tonalité lors d'un changement de
                                      ton
              AjouteBecarresApres(short, short);
        BOOLEAN                 //    Ajout d'un changement de tonalité
                                      courante dans la mesure
              AjouteChangementTon(short);
        BOOLEAN                 //    Ajout de l'indication de la mesure
                                      courante dans la mesure
              AjouteIndicationMesure();
        BOOLEAN AjouteCle();    //    Ajout de la clé courante dans la
                                      mesure
        BOOLEAN                 //    Ajout d'une barre de mesure dans
                                      la meusre
              AjouteBarreDeMesure();
        BOOLEAN AjouteNote      //    Ajout d'une note dans la mesure
              (CMusiqueNote *);
        int TraiteCodeSilence   //    Traitement d'un code de silence
              (const unsigned char *, int &);
        void TraiteCodeCle      //    Traitement d'un code de changement
                                      de clé
              (const unsigned char *) const;
        short TraiteCodeTon     //    Traitement d'un code de changement
                                      de ton
              (const unsigned char *);
        void TraiteCodeMesure   //    Traitement d'un code d'indication
                                      de mesure
              (const unsigned char *) const;
        BOOLEAN TraiteCodeNOlet //    Traitement d'un code de n_olet
              (const unsigned char *);
        BOOLEAN                 //    Traitement d'un code de liaison
              TraiteCodeLiaison(const unsigned char *) const;
        CMusiqueNote *          //    Traitement d'un code de note
              TraiteCodeNote(const unsigned char *);
        BOOLEAN TraiteBarreNote //    Ajout d'une barre de note ou des
                                      moustaches d'une note
              (int &, short &);
        BOOLEAN                 //    Ajout des barres de notes dans le
                                      brouillon
              AjouteSignetsBarresNotes();
        inline BOOLEAN          //    La mesure est terminée ou pas
              MesureIncomplete() const
              {return !((CGraphiqueMusical *)graphiques.last())->
               GraphiqueFinMesure();}
        short CalculeTemps      //    Calcul du temps de la mesure
              (short, short) const;

public:                         // Interface standard:
      CDecoupageMesure          //    Constructeur
```

```
            (CDecoupagePartition *, CDecoupagePortee *, long, long);
        ~CDecoupageMesure();    //    Destructeur
        int IndicePremiereNote  //    Calcul de l'indice de la première
                                     note de la mesure
            () const;
        int IndiceDerniereNote  //    Calcul de l'indice de la dernière
                                     note de la mesure
            () const;
        BOOLEAN Ajoute          //    Ajoute une mélodie à la mesure
            (const unsigned char *&, BOOLEAN &);
        void                    //    Translate tous les ordonnées des
                                     objets graphiques de la mesure
            TranslationVerticale(long);
        void ArrangeGraphiques  //    Réorganisation des graphiques en
                                     fonction des pixels accordés et du
                                     décalage subi
            (short, short);
        void                    //    Réinitialisation du vecteur
                                     d'objets graphiques de la mesure
            EffaceObjetsGraphiques();
        BOOLEAN                 //    Ajoute un continuateur de mesure à
                                     la fin de cette mesure
            AjouteContinuateur();
        BOOLEAN AjouteArmature  //    Ajoute les graphiques de
                                     l'armature courante dans la mesure
            (const unsigned char *&);
        BOOLEAN                 //    Ajout des graphiques de barres de
                                     notes
            AjouteBarresNotes();
        BOOLEAN AjouteNOlets(); //    Ajout des graphiques des n_olets
        inline                  //    Accès à la partition contenant la
                                     mesure
            CDecoupagePartition *Partition() {return partition;}
        inline                  //    Accès à la partition contenant la
                                     mesure
            CDecoupagePortee *Portee() {return portee;}
        inline CVecteur *       //    Accès à la liste des objets
                                     graphiques de la mesure
            Graphiques() {return &graphiques;}
        inline short            //    Accès au dépassement haut
            DepassementHaut() const {return depassementHaut;}
        inline short            //    Accès au dépassement bas
            DepassementBas() const {return depassementBas;}
};
#endif
```

CDecMesu.cpp

```
// Nature: Implémentation de la classe décrivant une mesure de découpage
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 24/10/96
// Modifications:
// Commentaires:
#ifdef MUSIQUE

#include "WaveDef.h"
#include CVecteurIterateur_i

#include "MusiDef.h"
#include CDecoupagePartition_i
#include CDecoupagePortee_i
#include CDecoupageMesure_i
#include CInitialisateurGraphique_i
#include CGraphiqueCles_i
```

```
#include CGraphiqueMesures_i
#include CGraphiqueAlterations_i
#include CGraphiqueAutres_i
#include CGraphiqueNotes_i
#include CGraphiqueSilences_i
#include CGraphiquePortee_i
#include CBrouillonGraphiqueHorizontal_i
#include ConstantesMusique_i

// Mise à jour des dépassements si un des paramètres
// est plus grand que les dépassements courants
void CDecoupageMesure::ChangeDepassements
      (short nouveauDepassementHaut, short nouveauDepassementBas)
{     if (depassementHaut <         // On met à jour le dépassement haut si
                                       le nouveau est plus grand que
                                       l'ancien
            nouveauDepassementHaut)
            depassementHaut = nouveauDepassementHaut;
      if (depassementBas <          // On met à jour le dépassement bas si
                                       le nouveau est plus grand que
                                       l'ancien
            nouveauDepassementBas)
            depassementBas = nouveauDepassementBas;
}

// Fonction privée d'ajout d'un graphique dans la mesure
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::AjouteGraphique
      (CGraphiqueMusical *graphique, short largeurGraphique,
       short espaceAvantGraphique, short espaceApresGraphique)
{     float echelle = partition->   // Echelle de redimensionnement des
                                       graphiques musicaux
            Echelle();
      CVecteur *objetsGraphiques    // Liste des objets graphiques de la
                                       partition
            = partition->ObjetsGraphiques();
      if (portee->                  // Test de la place disponible dans la
                                       portée
            LargeurDisponible() <
            (largeur + int((LARGEUR_CONTINUATEUR_MESURE + largeurGraphique
            + espaceAvantGraphique + espaceApresGraphique) * echelle)))
      {     delete graphique;
            return FALSE;
      }
      objetsGraphiques->            // On note le graphique dans la
                                       partition
           push(graphique);
      graphiques.push(graphique);   // On note le graphique dans la mesure
      largeur += int(               // Mise à jour de la largeur de la
                                       mesure
            (espaceAvantGraphique + largeurGraphique +
             espaceApresGraphique) * echelle);
      return TRUE;
}

// Fonction privée d'ajout de la tonalité dans la mesure
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::AjouteTonalite()
{     float echelle = partition->   // Echelle de redimensionnement des
                                       graphiques
            Echelle();
      CGraphiqueMusical *           // Objet graphique des altérations de
                                       tonalité
            alterationGraphique;
```

```
      short ton,                    // Nouvelle tonalité
            alteration = 0;         // Numéro de l'altération traitée
      const CMusiqueCle *cle;       // Clé courante
      ton = partition->Ton();       // On initialise le ton courant
      cle = partition->Cle();       // On initialise la clé courante
      if (ton == 0) return TRUE;    // Si on est en Do, on a termine avec
                                       succes
      if (ton < 0)                  // Si on est dans une tonalité à bémols
      {     while (--alteration >= ton)
            {     if (!             // On crée le nouvel objet graphique
                        (alterationGraphique =
                         new CGraphiqueBemol(positionX +
                         long(ESPACE_AVANT_BEMOL * echelle) + largeur,
                         int(cle->HauteurAlteration(alteration) * echelle)
                        ))) return FALSE;
                  if (!             // On note le bémol dans la mesure
                        AjouteGraphique(alterationGraphique, LARGEUR_BEMOL,
                              ESPACE_AVANT_BEMOL, ESPACE_APRES_BEMOL))
                        return FALSE;
                  if (              // On met à jour,la cas échéant, le
                                       dépassement au-dessus de la portée
                        depassementHaut < -cle->
                              HauteurAlteration(alteration))
                        depassementHaut = -cle->
                              HauteurAlteration(alteration);
            }
            largeur += int(         // On ajoute un espace après
                                       l'indication de la tonalité
                  ESPACE_APRES_TONALITE * echelle);
            return TRUE;
      }
      while (++alteration <= ton)   // On est dans une tonalité à dièses
      {     if (!(                  // On crée le nouvel objet graphique
                  alterationGraphique = new CGraphiqueDiese
                  (positionX +
                   long(ESPACE_AVANT_DIESE * echelle) + largeur,
                   int(cle->HauteurAlteration(alteration) * echelle), TRUE
                  ))) return FALSE;
            if (!AjouteGraphique    // On note le dièse dans la mesure
                  (alterationGraphique, LARGEUR_DIESE, ESPACE_AVANT_DIESE,
                   ESPACE_APRES_DIESE)) return FALSE;
            if (depassementHaut <   // On met à jour,la cas échéant, le
                                       dépassement au-dessus de la portée
                  -cle->HauteurAlteration(alteration))
                  depassementHaut = -cle->HauteurAlteration(alteration);
      }
      largeur += int(ESPACE_APRES_TONALITE * echelle);
      return TRUE;
}

// Fonction privée d'ajout d'un silence dans la mesure
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::AjouteSilence(int rythme, int rythmeVu)
{     float echelle = partition->   // Echelle de redimensionnement des
                                       graphiques musicaux
            Echelle();
                                  // Brouillon du graphique du silence
            CInitialisateurSilenceGraphique *brouillonSilence;
      if (!(brouillonSilence = new // On crée le brouillon du silence
            CInitialisateurSilenceGraphique(rythme, rythmeVu)
         )) return FALSE;
      if (!brouillonSilence->       // On initialise l'objet graphique du
                                       silence
            Initialise(echelle, positionX, largeur))
```

```
      {     delete brouillonSilence;
            return FALSE;
      }
      ChangeDepassements            // On met à jour les dépassements haut
                                       et bas de la mesure
            (brouillonSilence->DepassementHaut(),
             brouillonSilence->DepassementBas());
      BOOLEAN resultat =            // On note la note dans la mesure
            AjouteGraphique(brouillonSilence->Graphique(),
                            brouillonSilence->LargeurGraphique(),
                            brouillonSilence->EspaceAvant(),
                            brouillonSilence->EspaceApres());
      delete brouillonSilence;
      return resultat;
}

// Fonction privée d'ajout de bécarres avant la tonalité lors
// d'un changement de ton (on suppose donc que les tonalités ont le
// même signe)
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::AjouteBecarresApres
      (short ancienTon, short nouveauTon)
{     float echelle = partition->   // Echelle de redimensionnement des
                                       graphiques
            Echelle();
      CGraphiqueMusical *           // Objet graphique des altérations de
                                       tonalité
            alterationGraphique;
      short alterationCourante =    // Alteration à annuler par un bécarre
            nouveauTon;
      int ordonneeBecarre;          // Ordonnée du bécarre
      const CMusiqueCle *cle;       // Clé courante
      cle = partition->Cle();       // On initialise la clé courante
      if (nouveauTon < 0)           // Si on se trouve dans un ton bémolé
      {     while (                 // Tant que l'on n'a pas annuler toutes
                                       les altérations de l'ancien ton
                  --alterationCourante >= ancienTon)
            {     ordonneeBecarre = // On calcule l'ordonnée du bécarre
                        int(cle->HauteurAlteration(alterationCourante) +
                            HAUTEUR_BEMOL - 10 - HAUTEUR_BECARRE / 2);
                  if (!             // On crée le nouvel objet graphique
                        (alterationGraphique = new CGraphiqueBecarre
                         (positionX + long(ESPACE_AVANT_BECARRE * echelle)
                          + largeur, int(ordonneeBecarre * echelle))))
                        return FALSE;
                  if (!             // On note le bécarre dans la mesure
                        AjouteGraphique(alterationGraphique,
                              LARGEUR_BECARRE, ESPACE_AVANT_BECARRE,
                              ESPACE_APRES_BECARRE)) return FALSE;
                  ChangeDepassements// On met à jour,la cas échéant, les
                                       dépassements de la mesure
                        (-ordonneeBecarre, ordonneeBecarre - 48);
            }
            return TRUE;
      }
      while (++alterationCourante   // On était dans une tonalité à dièses
            <= ancienTon)
      {     ordonneeBecarre =       // On calcule l'ordonnée du bécarre
                  int(cle->HauteurAlteration(alterationCourante) +
                  (HAUTEUR_DIESE - HAUTEUR_BECARRE) / 2);
            if (!(                  // On crée le nouvel objet graphique
                  alterationGraphique = new CGraphiqueBecarre
                  (positionX + long(ESPACE_AVANT_DIESE * echelle) +
                   largeur, int(ordonneeBecarre * echelle)))) return FALSE;
```

```
            if (!AjouteGraphique    // On note le bécarre dans la mesure
                  (alterationGraphique, LARGEUR_BECARRE,
                   ESPACE_AVANT_BECARRE, ESPACE_APRES_BECARRE))
                  return FALSE;
            ChangeDepassements      // On met à jour,la cas échéant, les
                                       dépassements de la mesure
                  (-ordonneeBecarre, ordonneeBecarre - 48);
      }
      return TRUE;
}

// Fonction privée d'ajout de l'indication d'un changement de ton dans la
// mesure
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::AjouteChangementTon(short ancienTon)
{     float echelle = partition->   // Echelle de redimensionnement des
                                       graphiques
            Echelle();
      short nouveauTon;             // Nouvelle tonalité
      BOOLEAN nouveauTonBemol,      // drapeaux : les tons contiennent des
                                       bémols
              ancienTonBemol;
      short                         // Nombre de bémols ou de dièses de
                                       l'ancien et du nouveau ton
            nouveauNombreAlterations,
            ancienNombreAlterations;
      nouveauTon = partition->Ton();// On initialise le ton courant
      nouveauTonBemol = (nouveauTon < 0);
      ancienTonBemol = (ancienTon < 0);
      nouveauNombreAlterations =    // Initialisation des nombres
                                       d'altérations
            (nouveauTon > 0)? nouveauTon: -nouveauTon;
      ancienNombreAlterations =
            (ancienTon > 0)? ancienTon: -ancienTon;
      if (nouveauTon == ancienTon)  // Si les deux tons sont identiques, on
                                       ne fait rien
            return TRUE;
      if (nouveauTonBemol ==        // Si les deux tons sont soit dièses,
                                       soit bémol,
            ancienTonBemol)
      {     if (!AjouteTonalite())  // on ajoute la tonalité à la mesure
                  return FALSE;
            largeur -=              // On n'a pas encore fini le bloc de
                                       changement de ton
                  int(ESPACE_APRES_TONALITE * echelle);
            if (                    // et si on a moins d'altérations dans
                                       le nouveau ton que dans l'ancien,
                  nouveauNombreAlterations < ancienNombreAlterations)
                  if (!             // on ajoute les bécarres pour les
                                       altérations en moins
                        AjouteBecarresApres(ancienTon, nouveauTon))
                        return FALSE;
            largeur +=              // on ajoute l'espace après l'indication
                                       de tonalité
                  int(ESPACE_APRES_TONALITE * echelle);
            return TRUE;
      }
      else                          // Si un ton fait intervenir des bémols
                                       et l'autre des dièses
      return AjouteTonalite();      // et on ajoute la tonalité
}

// Fonction privée d'ajout de l'indication de la mesure dans la mesure
// Renvoie TRUE si tout c'est bien passé
```

```
BOOLEAN CDecoupageMesure::AjouteIndicationMesure()
{     BOOLEAN resumee                  // Drapeau : la mesure courante est sous
                                          forme résumée
            = partition->Mesure()->Resumee();
      short numerateur                 // Numérateur de la mesure courante
            = partition->Mesure()->Numerateur();
      short denominateur               // Dénominateur de la mesure courante
            = partition->Mesure()->Denominateur();
      float echelle = partition->      // Echelle de redimensionnement des
                                          graphiques musicaux
            Echelle();
      short largeurNumerateur,         // Largeur en pixel du numérateur
            largeurDenominateur;       // et du dénominateur
      char buffer[8];                  // Buffer de conversion des chiffres en
                                          ascii
      WINDOW fenetre = partition->     // Fenêtre d'écriture de la partition
            Partition()->Fenetre();
      CGraphiqueMusical *              // Graphique de la mesure à afficher
            mesureGraphique;
      if ((resumee) && (numerateur     // Cas de la mesure C
            == 4) && (denominateur == 4))
      {     if (!(mesureGraphique = // On crée le nouvel objet graphique
                  new CGraphiqueMesureC(positionX +
                        long(ESPACE_AVANT_MESURE * echelle) + largeur,
                                    // 12 : hauteur du graphique par rapport
                                       à la portée
                        long(12 * echelle)))) return FALSE;
            if (!AjouteGraphique    // On note la clé dans la mesure
                  (mesureGraphique, LARGEUR_C, ESPACE_AVANT_CLE,
                   ESPACE_APRES_MESURE)) return FALSE;
            return TRUE;
      }
      if ((resumee) && (numerateur     // Cas de la mesure C barré
            == 2) && (denominateur == 2))
      {     if (!(mesureGraphique = // On crée le nouvel objet graphique
                  new CGraphiqueMesureCBarre(positionX +
                        long(ESPACE_AVANT_MESURE * echelle) + largeur,
                                    // 4 : hauteur du graphique par rapport
                                       à la portée
                        long(4 * echelle)))) return FALSE;
            if (!AjouteGraphique    // On note la clé dans la mesure
                  (mesureGraphique, LARGEUR_CBARRE, ESPACE_AVANT_MESURE,
                   ESPACE_APRES_MESURE)) return FALSE;
            return TRUE;
      }
      sprintf(buffer, "%d",            // On convertit le numérateur en ascii
            numerateur);
      if (resumee)                     // Cas général d'une mesure sous forme
                                          résumée
      {     XVT_FNTID police           // Initialisation de la police grasse
                  = xvt_font_create();
            xvt_font_set_family(police, XVT_FFN_TIMES);
            xvt_font_set_size(police, long(echelle * 24));
            xvt_font_set_style(police, XVT_FS_BOLD);
            xvt_dwin_set_font(fenetre, police);
            largeurNumerateur =        // On calcule la largeur du numérateur
                  xvt_dwin_get_text_width(fenetre, buffer, -1);
            if (!(mesureGraphique = // On crée le nouvel objet graphique
                  new CGraphiqueMesureResumee
                        (numerateur, denominateur, positionX +
                         long(ESPACE_AVANT_MESURE * echelle) + largeur,
                                    // 12 : hauteur du graphique par rapport
                                       à la portée
                         long(12 * echelle))))
```

```
            {     xvt_font_destroy(police);
                  return FALSE;
            }
            if (!AjouteGraphique    // On note la clé dans la mesure
                  (mesureGraphique, largeurNumerateur, ESPACE_AVANT_MESURE,
                   ESPACE_APRES_MESURE))
            {     xvt_font_destroy(police);
                  return FALSE;
            }
            xvt_font_destroy(police);
            return TRUE;
      }
      XVT_FNTID police =            // Initialisation de la police grasse
            xvt_font_create();
      xvt_font_set_family(police, XVT_FFN_TIMES);
      xvt_font_set_size(police, long(echelle * 24));
      xvt_font_set_style(police, XVT_FS_NONE);
      xvt_dwin_set_font(fenetre, police);
      largeurNumerateur =           // On calcule la largeur du numérateur
            xvt_dwin_get_text_width(fenetre, buffer, -1);
      sprintf(buffer, "%d",         // On convertit le dénominateur en ascii
            denominateur);
      largeurDenominateur =         // On calcule la largeur du dénominateur
            xvt_dwin_get_text_width(fenetre, buffer, -1);
      if (!(mesureGraphique =       // On crée le nouvel objet graphique
            new CGraphiqueMesure(numerateur, denominateur, largeur +
                  positionX + long(ESPACE_AVANT_MESURE * echelle), 0)))
      {     xvt_font_destroy(police);
            return FALSE;
      }
      if (!AjouteGraphique          // On note la clé dans la mesure
            (mesureGraphique,
             (largeurNumerateur > largeurDenominateur)?
             largeurNumerateur:
             largeurDenominateur,
             ESPACE_AVANT_MESURE, ESPACE_APRES_MESURE))
      {     xvt_font_destroy(police);
            return FALSE;
      }
      xvt_font_destroy(police);
      return TRUE;
}

// Fonction privée d'ajout de la clé courante dans la mesure
// Renvoie FALSE si cela n'a pas été possible.
BOOLEAN CDecoupageMesure::AjouteCle()
{     float echelle = partition->   // Echelle de redimensionnement des
                                       graphiques musicaux
            Echelle();
      const CMusiqueCle *cle =      // Clé courante de la partition (clé à
                                       ajouter)
            partition->Cle();
      CGraphiqueMusical *           // Graphique de la clé à afficher
            cleGraphique;
      switch (cle->Nom())
      {     case cleFa:             // Cas de la clé de fa
                  if (!(            // On crée le nouvel objet graphique
                        cleGraphique = new CGraphiqueCleFa
                              (largeur + positionX +
                               long(ESPACE_AVANT_CLE * echelle),
                               long((60 - cle->Ligne() * 12) * echelle))))
                        return FALSE;
                  if (!             // On note la clé dans la mesure
                        AjouteGraphique(cleGraphique, LARGEUR_CLE_FA,
```

```
                              ESPACE_AVANT_CLE, ESPACE_APRES_CLE))
                        return FALSE;
                  return TRUE;
            case cleUt:             // Cas de la clé d'ut
                  if (!(            // On crée le nouvel objet graphique
                        cleGraphique = new CGraphiqueCleUt(largeur +
                              positionX + long(ESPACE_AVANT_CLE * echelle),
                              long((36 - cle->Ligne() * 12) * echelle))))
                        return FALSE;
                  if (!             // On note la clé dans la mesure
                        AjouteGraphique(cleGraphique, LARGEUR_CLE_UT,
                              ESPACE_AVANT_CLE, ESPACE_APRES_CLE))
                        return FALSE;
                  ChangeDepassements// On change les dépassements haut et
                                    bas si c'est nécessaire
                        ((cle->Ligne() - 3) * 12,
                         -(cle->Ligne() - 3) * 12);
                  return TRUE;
            case cleSol:
                  if (!(            // On crée le nouvel objet graphique
                        cleGraphique = new CGraphiqueCleSol
                              (largeur + positionX +
                               long(ESPACE_AVANT_CLE * echelle),
                               long((6 - cle->Ligne() * 12) * echelle))))
                        return FALSE;
                  if (!             // On note la clé dans la mesure
                        AjouteGraphique(cleGraphique, LARGEUR_CLE_SOL,
                              ESPACE_AVANT_CLE, ESPACE_APRES_CLE))
                        return FALSE;
                  ChangeDepassements// On change les dépassements haut et
                                    bas si c'est nécessaire
                        (-6 + cle->Ligne() * 12,
                         (cle->Ligne() == 1)? 32: 20);
                  return TRUE;
      }
      return FALSE;
}

// Fonction privée d'ajout d'une barre de mesure dans la mesure
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::AjouteBarreDeMesure()
{     float echelle = partition->   // Echelle de redimensionnement des
                                    graphiques musicaux
            Echelle();
      CVecteur *objetsGraphiques    // Liste des objets graphiques de la
                                    partition
            = partition->ObjetsGraphiques();
      CGraphiqueMusical *           // Objet graphique de la barre de mesure
            barreMesure;
      if (portee->                  // Test de la place disponible dans la
                                    portée
            LargeurDisponible() - largeur <
            short((ESPACE_AVANT_BARRE_MESURE + ESPACE_APRES_BARRE_MESURE)
            * echelle))
            return FALSE;
      if (!(barreMesure = new       // On crée le nouvel objet graphique
            CGraphiqueBarreDeMesure(positionX + largeur +
                  long(ESPACE_AVANT_BARRE_MESURE * echelle), 0L)))
            return FALSE;
      objetsGraphiques->push        // On le note dans la partition
            (barreMesure);
      graphiques.push(barreMesure); // On le note dans la mesure
      largeur += 1 +                // Mise à jour de la largeur de la
                                    mesure
```

```
            short((ESPACE_AVANT_BARRE_MESURE + ESPACE_APRES_BARRE_MESURE)
            * echelle);
      return TRUE;
}

// Fonction privée d'ajout d'une note dans la mesure
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::AjouteNote(CMusiqueNote *note)
{     float echelle = partition->   // Echelle de redimensionnement des
                                      graphiques musicaux
            Echelle();
      CMusiqueCle *cleCourante      // Clé courante de la partition
            = partition->Cle();
      CMusiqueHauteurNote *         // Hauteur de la note en son équivalent
                                      (visuel) clé de sol
            hauteurVue;
      short ordonnee,               // Ordonnée de la note par rapport à la
                                      première ligne de la portée
            nombreLignes;
      CInitialisateurAlterationGraphique *brouillonAlteration;
                                    // Brouillon d'initialisation de
                                      l'altération accidentelle éventuelle
      CInitialisateurNoteGraphique  // Brouillon d'initialisation de la note
            *brouillonNote;
      hauteurVue = note->Hauteur(); // On initialise la hauteur vue à la
                                      hauteur réelle
      hauteurVue->                  // On transpose la note en son
                                      équivalent (visuel) en clé de sol
                                      2ème ligne
            TransposeCleSol2(cleCourante);
      nombreLignes = hauteurVue->   // On calcule les paramètres de lignes
                                      supplémentaires de la note
            CalculeLignesSupplementaires();
      ordonnee = hauteurVue->       // On calcule l'ordonnée du point
                                      supérieur gauche du corps de la note
            CalculeOrdonnee();
      if (note->Alteration() !=     // Si on a une altération accidentelle
            alterationsCourantes[note->Hauteur()->Note()])
      {     alterationsCourantes    // On ajoute cette altération
                                      accidentelle à la mesure
                  [note->Hauteur()->Note()] = note->Alteration();
            if (!(                  // On crée le brouillon de l'altération
                                      accidentelle
                  brouillonAlteration = new
                        CInitialisateurAlterationGraphique
                        (ordonnee, note->Alteration())
               )) return FALSE;
            if (!brouillonAlteration// On initialise l'objet graphique de
                                    l'altération accidentelle
                  ->Initialise(echelle, positionX, largeur))
            {     delete brouillonAlteration;
                  return FALSE;
            }
            ChangeDepassements      // On met à jour les dépassements haut
                                      et bas de la mesure
                  (brouillonAlteration->DepassementHaut(),
                   brouillonAlteration->DepassementBas());
            if (!AjouteGraphique    // On note l'altération dans la mesure
                  (brouillonAlteration->Graphique(),
                   brouillonAlteration->LargeurGraphique(),
                   brouillonAlteration->EspaceAvant(),
                   brouillonAlteration->EspaceApres()
               ))
            {     delete brouillonAlteration;
```

```
                  return FALSE;
            }
            delete brouillonAlteration;
      }
      if (!(brouillonNote = new     // On crée le brouillon de l'altération
                                       accidentelle
            CInitialisateurNoteGraphique
                  (ordonnee, note->Duree(), note->RythmeVu(), nombreLignes)
         )) return FALSE;
      if (!brouillonNote->Initialise// On initialise l'objet graphique de
                                       l'altération accidentelle
            (echelle, positionX, largeur))
      {     delete brouillonNote;
            return FALSE;
      }
      ChangeDepassements            // On met à jour les dépassements haut
                                       et bas de la mesure
            (brouillonNote->DepassementHaut(),
             brouillonNote->DepassementBas());
      BOOLEAN resultat =            // On note la note dans la mesure
            AjouteGraphique(brouillonNote->Graphique(),
                  brouillonNote->LargeurGraphique(),
                  brouillonNote->EspaceAvant(),
                  brouillonNote->EspaceApres());
      delete brouillonNote;
      return resultat;
}

// Fonction privée de traitement d'un code de silence
// Renvoie le rythme du silence correspondant au code
int CDecoupageMesure::TraiteCodeSilence
      (const unsigned char *codeNote, int &rythmeVu)
{     int rythme =                  // Rythme du silence
            (*++codeNote & 0x0Fu) << 8;
      rythme += *++codeNote;
      if (nombreNotesNOlet)         // Si on se trouve dans un n-olet,
      {     rythmeVu = int(rythme * // Le rythme vu est différent du rythme
                                       joué
                  ((CBrouillonGraphiqueNOlet *) brouillonNOlets.last()
                   )->CoefficientRythmique());
      }
      else rythmeVu = rythme;
      return rythme;
}

// Fonction privée de traitement d'un code de changement de clé
void CDecoupageMesure::TraiteCodeCle(const unsigned char *codeCle) const
{     short ligneCle;               // Ligne de la nouvelle clé
      nomCle nom;                   // Nom de la nouvelle clé
      codeCle++;                    // On passe le premier octet du code
      ligneCle =(*++codeCle &       // On calcule la ligne de la nouvelle
                                       clé
            '\x0F');
      switch (*codeCle >> 4)        // On change le type de la clé courante
      {     case 2: nom = cleUt;
                  break;
            case 3: nom = cleFa;
                  break;
            default: nom = cleSol;
      }
      partition->Cle(nom, ligneCle);// On change la clé
}
```

```
// Fonction privée de traitement d'un code de changement de ton
// Renvoie le ton précédent
short CDecoupageMesure::TraiteCodeTon(const unsigned char *codeTon)
{     short ancienTon = partition   // On note l'ancienne tonalité
            ->Ton();
      short *noteCourante;          // Note dans le tableau des altérations
                                       courantes
      codeTon += 2;                 // On se positionne sur le dernier octet
                                       du code
      if (*codeTon <=7)
            partition->Ton          // On change le ton courant si le ton
                                       est avec des dièses
            (*codeTon);
      else partition->Ton           // On fait la même chose si le ton est
                                       avec des bémols
            (*codeTon - 16);
      for (noteCourante = alterationsCourantes;
           noteCourante < alterationsCourantes + 7;
           *noteCourante++ = 0);
      switch (partition->Ton())     // On réinitialise le tableau des
                                       altérations courantes
      {     case -7:                // fa bémol
                  alterationsCourantes[3] = -1;
            case -6:                // do bémol
                  alterationsCourantes[0] = -1;
            case -5:                // sol bémol
                  alterationsCourantes[4] = -1;
            case -4:                // ré bémol
                  alterationsCourantes[1] = -1;
            case -3:                // la bémol
                  alterationsCourantes[5] = -1;
            case -2:                // mi bémol
                  alterationsCourantes[2] = -1;
            case -1:                // si bémol
                  alterationsCourantes[6] = -1;
                  break;
            case 7:                 // si dièse
                  alterationsCourantes[6] = 1;
            case 6:                 // mi dièse
                  alterationsCourantes[2] = 1;
            case 5:                 // la dièse
                  alterationsCourantes[5] = 1;
            case 4:                 // ré bémol
                  alterationsCourantes[1] = 1;
            case 3:                 // sol dièse
                  alterationsCourantes[4] = 1;
            case 2:                 // do dièse
                  alterationsCourantes[0] = 1;
            case 1:                 // fa dièse
                  alterationsCourantes[3] = 1;
      }
      return ancienTon;
}

// Fonction privée de traitement d'un code d'indication de mesure
void CDecoupageMesure::TraiteCodeMesure(const unsigned char *codeMesure)
const
{     partition->Mesure()->Resumee  // On met à jour le drapeau d'indication
                                       de mesure résumée
            (*codeMesure & 0x01u);
      partition->Mesure()->         // On met à jour le numérateur
            Numerateur(*++codeMesure);
      partition->Mesure()->         // On met à jour le dénominateur
            Denominateur(*++codeMesure);
```

```
}

// Fonction privée de traitement d'un code de n-olet
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::TraiteCodeNOlet
      (const unsigned char *codeNOlet)
{     short n = *++codeNOlet,       // N du n-olet
            nombreNotes =           // Nombre de notes entrant dans ce n-
                                       olet
            *++codeNOlet,
            denominateur;           // Dénominateur du coefficient rythmique
                                       du n-olet
      CBrouillonGraphiqueNOlet *    // N-olet à insérer
            nOlet;
      denominateur = n >> 4;        // On calcule le dénominateur du
                                       coefficient rythmique
      n = n & 0x0fu;                // On calcule le n
      if (!denominateur)            // Si le dénominateur est nul il y a
                                       erreur
            return FALSE;
      if (!(nOlet =                 // On crée le n-olet
            new CBrouillonGraphiqueNOlet(n, float(n) / float(denominateur),
                  graphiques.entries(), nombreNotes))) return FALSE;
      brouillonNOlets.push(nOlet);  // On note ce n-olet dans la liste de n-
                                       olet de la mesure
      nombreNotesNOlet =            // On met à jour le nombre de note du n-
                                       olet courant
            nombreNotes;
      return TRUE;
}

// Fonction privée de traitement d'un code de liaison
// Renvoie TRUE si tout s'est bien passé
BOOLEAN CDecoupageMesure::TraiteCodeLiaison
      (const unsigned char *codeLiaison) const
{     float echelle = partition->   // Echelle de redimensionnement des
                                       graphiques musicaux
            Echelle();
      CVecteur *brouillonLiaisons   // Vecteur des liaisons de la partition
            = partition->BrouillonLiaisons();
      CVecteur *graphiques =        // Vecteur des objets graphiques
                                       musicaux de la partition
            partition->ObjetsGraphiques();
      short nombreNotes =           // Nombre de notes entrant dans cette
                                       liaison
            *++codeLiaison;
      CBrouillonGraphiqueLiaison *  // Liaison à insérer
            liaison;
      if (!(liaison =               // On crée la liaison
            new CBrouillonGraphiqueLiaison(partition->ObjetsGraphiques()
                  ->entries(), nombreNotes, graphiques, echelle)))
            return FALSE;
      brouillonLiaisons->push       // On note cette liaison dans la liste
                                       des liaisons de la partition
            (liaison);
      return TRUE;
}

// Fonction privée de traitement d'un code de note
// Renvoie la note correspondant au code ou NULL en cas d'erreur
// L'appelant est responsable de la destruction de cette note
CMusiqueNote *CDecoupageMesure::TraiteCodeNote
      (const unsigned char *codeNote)
{     CMusiqueHauteurNote           // Hauteur de la note
```

```
      hauteur(-2, CMusiqueHauteurNote::Do);
CMusiqueNote::alterateur      // Altération de la note
      alteration(CMusiqueNote::Becarre);
int hauteurCourante = 0;      // Compteur de demi-tons
int codeHauteur = *codeNote;  // Hauteur codée de la note
short codeAlteration          // Altération codée de la note
      = (*++codeNote & 0xF0u) >> 6;
int duree = (*codeNote        // Duree de la note
      & 0x0Fu) << 8;
int rythmeVu;                 // Rythme vu dans la partition
duree += *++codeNote;
if (nombreNotesNOlet)         // Si on se trouve dans un n-olet,
{     rythmeVu = int(duree *  // Le rythme vu est différent du rythme
                              joué
            ((CBrouillonGraphiqueNOlet *) brouillonNOlets.last())->
            CoefficientRythmique());
}
else rythmeVu = duree;
while (hauteurCourante++ <    // On cherche la hauteur et l'octave de
                              la note
      codeHauteur)
{     if (alteration          // Si la note est Si ou Mi ou si la note
                              est dièse,
            || (hauteur.Note() == CMusiqueHauteurNote::Si)
            || (hauteur.Note() == CMusiqueHauteurNote::Mi))
      {     hauteur.          // on augmente la hauteur de la note
                  Transpose(1);
            alteration = CMusiqueNote::Becarre;
      }
      else                    // Dans les autre cas, la note devient
                              dièse
      {     alteration = CMusiqueNote::Diese;
      }
}
switch (codeAlteration)
{     case 1:                 // Cas bémol ou double bémol
            if (!alteration   // Si la note n'est ni un Si, ni un Mi
                              et n'a pas d'altération, il s'agit de
                              la note au-dessus avec un double
                              bémol
                  && (hauteur.Note() != CMusiqueHauteurNote::Mi)
                  && (hauteur.Note() != CMusiqueHauteurNote::Si))
            {     alteration = CMusiqueNote::DoubleBemol;
            }
            else              // Sinon, il s'agit de la note au-dessus
                              avec un bémol
            {     alteration = CMusiqueNote::Bemol;
            }
            hauteur.Transpose(1);
            break;
      case 3:                 // Cas dièse ou double dièse
            if (alteration)   // Si la note a une altération, le dièse
                              y est déjà
                  break;
                              // Sinon, il s'agit de la note en-
                              dessous avec un double dièse
            if (   (hauteur.Note() != CMusiqueHauteurNote::Fa)
                && (hauteur.Note() != CMusiqueHauteurNote::Do))
                  alteration = CMusiqueNote::DoubleDiese;
            else              // Sauf si la note est Fa ou Do, il
                              s'agit alors de la note en-dessous
                              avec un dièse
                  alteration = CMusiqueNote::Diese;
            hauteur.Transpose(-1);
```

```
                  break;
            }
            return new CMusiqueNote       // On renvoie la note correspondant au
                                             code traité
                  (&hauteur, alteration, duree, rythmeVu);
}

// Fonction privée d'ajout d'une barre de note ou des moustaches d'une note
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::TraiteBarreNote
      (int &indiceGraphiqueCourant, short &tempsRestant)
{     float echelle = partition->   // Echelle de redimensionnement des
                                       graphiques musicaux
            Echelle();
      CBrouillonGraphiqueBarre *    // Barre à ajouter au vecteur de
                                       brouillon
            barre;
      CMusiqueMesure *mesure        // Indication de mesure employée dans
                                       cette mesure
            = partition->Mesure();
      CGraphiqueNote *noteOrpheline;// Note d'une barre de niveau 1 qui ne
                                       contient qu'une note
      short depassementHautNote = 0,// Dépassements haut et bas d'une note
                                       orpheline
            depassementBasNote = 0;
      short temps = CalculeTemps    // Nombre de 128ième de noire pour un
                                       temps
            (mesure->Numerateur(), mesure->Denominateur());
      noteOrpheline =               // On se souvient de la note courante
            (CGraphiqueNote *) graphiques.at(indiceGraphiqueCourant);
      if (!(barre = new             // On crée une nouvelle barre de niveau1
            CBrouillonGraphiqueBarre(indiceGraphiqueCourant, 1)))
            return FALSE;
      if (!barre->AjouteGraphiques  // On ajoute des graphiques à cette
                                       barre
            (this, indiceGraphiqueCourant, tempsRestant))
            return FALSE;
      if (barre->NombreNotes() <= 1)// Si on a une barre de niveau 1 à une
                                       seule note,
      {     noteOrpheline->         // on ajoute les moustaches graphiques à
                                       la seule note de cette mesure
                  AjouteMoustachesGraphiques(echelle);
            depassementHautNote =   // On note les dépassements haut et bas
                                       de la note
                  short(-noteOrpheline->OrdonneeMinimale(echelle) /
                  echelle);
            depassementBasNote =
                  short(noteOrpheline->OrdonneeMaximale(echelle) / echelle)
                  - (HAUTEUR_INTERLIGNE << 2);
            ChangeDepassements      // On change les dépassements haut et
                                       bas de la mesure si c'est nécessaire
                  (depassementHautNote, depassementBasNote);
            delete barre;           // on supprime la barre,
            return TRUE;
      }                             // Si on a plus d'une note dans la barre
      brouillonBarresNotes.push     // et on note cette barre
            (barre);
      return TRUE;
}

// Fonction privée d'ajout des signets de barres de notes
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::AjouteSignetsBarresNotes()
{     CGraphiqueMusical *           // Objet graphique de la mesure
```

```
            graphiqueCourant;
      CMusiqueMesure *mesure        // Indication de mesure employée dans
                                       cette mesure
            = partition->Mesure();
      short temps,                  // Nombre de 128ième de noire pour un
                                       temps
            tempsRestant;           // Temps restant dans une barre donnée
      int indiceGraphiqueCourant    // Indice du graphique courant
            = 0;
      temps = CalculeTemps          // On calcule le temps
            (mesure->Numerateur(), mesure->Denominateur());
      tempsRestant = temps;         // On initialise le temps restant
      while (indiceGraphiqueCourant // Tant que l'on n'a pas traiter tous
                                       les graphiques de la mesure
            < graphiques.entries())
      {     graphiqueCourant =      // On va chercher le graphique courant
                  (CGraphiqueMusical *) graphiques.at
                        (indiceGraphiqueCourant);
            if (graphiqueCourant-> // Si ce graphique est une note
                  Note())
            {     if (((           // Si on a une note sans moustache,
                        CGraphiqueNote *) graphiqueCourant)->
                        NombreMoustaches() == 0)
                  {     tempsRestant// On diminue le temps restant
                              -= ((CGraphiqueNote *) graphiqueCourant)->
                                    Duree();
                  }
                  else if (!        // on ajoute soit une barre, soit des
                                       moustaches à la note
                        TraiteBarreNote(indiceGraphiqueCourant,
                              tempsRestant)) return FALSE;
            }
            else if (graphiqueCourant->Silence())
            {     tempsRestant -=   // On diminue le temps restant
                        ((CGraphiqueSilence *) graphiqueCourant)->Duree();
            }                       // On ignore les graphiques qui ne sont
ni
                                       des notes, ni des silences
            while (tempsRestant     // Si le temps restant est négatif, on
                                       lui ajoute assez de temps pour qu'il
                                       redevienne positif
                  <= 0) tempsRestant += temps;
                                    // On incrémente l'indice du graphique
                                       courant
            indiceGraphiqueCourant++;
      }
      return TRUE;
}

// Calcul du temps de la mesure
short CDecoupageMesure::CalculeTemps
      (short numerateur, short denominateur) const
{     return ((numerateur % 3) || (numerateur == 3 && denominateur <= 4))?
            384 / denominateur:     // Pour une mesure binaire
            1152 / denominateur;    // Pour une mesure ternaire
}

// Constructeur
CDecoupageMesure::CDecoupageMesure
      (CDecoupagePartition *partitionDecoupage, CDecoupagePortee
*porteeDecoupage,
       long abscisse, long ordonnee)
      : CDecoupageMusical(abscisse, ordonnee, 0), graphiques(8),
        brouillonNOlets(4), brouillonBarresNotes(8)
```

```
{     short *noteCourante;          // Note dans le tableau des altérations
                                       courantes
      partition =                   // On Rnitialise la partition
            partitionDecoupage;
      portee = porteeDecoupage;     // et la portée dans lesquelles s'insère
                                       la mesure
      depassementHaut =             // On commence avec une mesure vierge
            depassementBas = 0;
      nombreNotesNOlet = 0;         // On n'est pas dans un n-olet
      for (noteCourante = alterationsCourantes;
           noteCourante < alterationsCourantes + 7;
           *noteCourante++ = 0);
      switch (partition->Ton())     // On initialise le tableau des
                                       altérations courantes
      {     case -7:                // fa bémol
                  alterationsCourantes[3] = -1;
            case -6:                // do bémol
                  alterationsCourantes[0] = -1;
            case -5:                // sol bémol
                  alterationsCourantes[4] = -1;
            case -4:                // ré bémol
                  alterationsCourantes[1] = -1;
            case -3:                // la bémol
                  alterationsCourantes[5] = -1;
            case -2:                // mi bémol
                  alterationsCourantes[2] = -1;
            case -1:                // si bémol
                  alterationsCourantes[6] = -1;
                  break;
            case 7:                 // si dièse
                  alterationsCourantes[6] = 1;
            case 6:                 // mi dièse
                  alterationsCourantes[2] = 1;
            case 5:                 // la dièse
                  alterationsCourantes[5] = 1;
            case 4:                 // ré bémol
                  alterationsCourantes[1] = 1;
            case 3:                 // sol dièse
                  alterationsCourantes[4] = 1;
            case 2:                 // do dièse
                  alterationsCourantes[0] = 1;
            case 1:                 // fa dièse
                  alterationsCourantes[3] = 1;
      }
}

// Destructeur
CDecoupageMesure::~CDecoupageMesure()
{     brouillonNOlets.clearAndDestroy();
      brouillonBarresNotes.clearAndDestroy();
}

// Calcul de l'indice de la première note de la mesure
// Renvoie 0 s'il n'y a pas de note
int CDecoupageMesure::IndicePremiereNote() const
{     CVecteurIterateur             // Itérateur sur les objets graphiques
                                       de la mesure
            graphiqueMesureSuivant(graphiques);
      CVecteurIterateur             // Itérateur sur les objets graphiques
                                       de la partition
            graphiqueSuivant(*(partition->ObjetsGraphiques()));
      CGraphiqueMusical             // Première note de la mesure
                  *premiereNote,
                  *note;            // Note courante dans le vecteur des
```

```
                                    graphiques de la partition
      int indice = 0;               // Indice du graphique courant
      while (premiereNote =         // On avance jusqu'à la première note de
                                    la mesure
            ((CGraphiqueMusical *) graphiqueMesureSuivant()))
            if (premiereNote->      // On a trouvé la première note de la
                                    mesure
                  Note()) break;
      if (!premiereNote) return 0;  // Il n'y a pas de note dans cette
                                    mesure
      while (note =                 // On cherche l'indice dans le vecteur
                                    des graphiques de la partition
            (CGraphiqueMusical *) graphiqueSuivant())
      {     if (note ==             // On a trouvé la première note dans le
                                    vecteur
                  premiereNote) return indice;
            indice++;               // On incrémente l'indice
      }
      return 0;                     // Il y a eu un problème !!
}

// Calcul de l'indice de la dernière note de la mesure
// Renvoie 0 s'il n'y a pas de note
int CDecoupageMesure::IndiceDerniereNote() const
{     CVecteurIterateur             // Itérateur sur les objets graphiques
                                    de la mesure
            graphiqueMesureSuivant(graphiques);
      CVecteurIterateur             // Itérateur sur les objets graphiques
                                    de la partition
            graphiqueSuivant(*(partition->ObjetsGraphiques()));
      CGraphiqueMusical             // Première note de la mesure
            *derniereNote,
            *note,                  // Note courante dans le vecteur des
                                    graphiques de la partition
            *uneNote;               // Note intermédiaire de la mesure
      int indice = 0;               // Indice du graphique courant
      dernieNote = NULL;
      while (uneNote = (            // On avance jusqu'à la première note de
                                    la mesure
            (CGraphiqueMusical *) graphiqueMesureSuivant()))
            if (uneNote->Note())    // On a trouvé une note de la mesure
                  dernieNote =      // On dit que c'est la dernière
                        uneNote;
      if (!dernieNote) return 0;    // Il n'y a pas de note dans cette
                                    mesure
      while (note =                 // On cherche l'indice dans le vecteur
                                    des graphiques de la partition
            (CGraphiqueMusical *) graphiqueSuivant())
      {     if (note ==             // On a trouvé la première note dans le
                                    vecteur
                  dernieNote) return indice;
            indice++;               // On incrémente l'indice
      }
      return 0;                     // Il y a eu un problème !!
}

// Ajout d'une melodie dans une mesure
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::Ajoute
      (const unsigned char *&melodie, BOOLEAN &mesureComplete)
{     const unsigned char *         // Pointeur vers la mélodie
            codeATraiter = melodie;
      CMusiqueNote *note;           // Note traitée
      short sauvegardeTon =         // Copie du ton courant avant le
```

```
                                    traitement de cette mesure
      partition->Ton();
int rythme,                         // Rythme d'un silence
    rythmeVu;
BOOLEAN mesurePleine = FALSE; // Initialisation de l'état de
                                    remplissage de la mesure
mesureComplete = FALSE;             // La mesure n'est pas complète
while (((*codeATraiter & 0xF0u) != 0xF0u) && !mesurePleine)
{     switch (*codeATraiter & 0xF0u)
      {     case 0x80u:             // Cas du silence
                  rythme =          // On traite le code
                        TraiteCodeSilence(codeATraiter, rythmeVu);
                  if (!             // On ajoute la note dans la mesure
                        AjouteSilence(rythme, rythmeVu))
                        mesurePleine = TRUE;
                  if (nombreNotesNOlet) nombreNotesNOlet--;
                  break;
            case 0x90u:             // Cas des changements de clé ou de ton
                                    // Cas du changement de ton
                  if (!(*codeATraiter & 0x0Fu))
                  {     if (!AjouteChangementTon
                              (TraiteCodeTon(codeATraiter)))
                              mesurePleine = TRUE;
                  }
                  else              // Cas du changement de clé
                  {     TraiteCodeCle(codeATraiter);
                        if (!AjouteCle()) mesurePleine = TRUE;
                  }
                  break;
            case 0xA0u:             // Cas de l'indication de mesure
                                    // On traite ce code
                  TraiteCodeMesure(codeATraiter);
                  if (!             // On ajoute l'indication de mesure
                        AjouteIndicationMesure()) mesurePleine = TRUE;
                  break;
            case 0xB0u:             // Cas de la barre de mesure
                  if (!             // On ajoute une barre de mesure à la
                                       mesure
                        AjouteBarreDeMesure()) mesurePleine = TRUE;
                                    // On passe ce code
                  codeATraiter += 3;
                  melodie =         // On avance dans la mélodie
                        codeATraiter;
                                    // On ajoute les barres de notes dans
                                       leur brouillon
                  if (!AjouteSignetsBarresNotes()) return FALSE;
                                    // La mesure est complète
                  mesureComplete = TRUE;
                  return TRUE;// On a traité avec succès cette mesure
            case 0xD0u:             // Cas du n-olet
                  if (!TraiteCodeNOlet(codeATraiter)) return FALSE;
                  break;
            case 0xE0u:             // Cas de la liaison
                  if (!TraiteCodeLiaison(codeATraiter)) return FALSE;
                  break;
            default:                // Cas des notes
                  if (!(note =// On traite le code de la note
                        TraiteCodeNote(codeATraiter)))
                  {     delete note;
                        return FALSE;
                  }
                  if (!             // On ajoute la note dans la mesure
                       AjouteNote(note)) mesurePleine = TRUE;
                  delete note;// On n'a plus besoin de la note
```

```
                        if (nombreNotesNOlet) nombreNotesNOlet--;
            }
            if (!mesurePleine) codeATraiter += 3;
      }
      if (codeATraiter == melodie)  // Si on n'a pas pu traiter un seul
                                       code, on n'a pas assez d'espace
      {     partition->Ton          // on restaure le ton avant traitement
                                       des codes de la mesure
                  (sauvegardeTon);
            return TRUE;
      }
      if ((mesurePleine) &&         // Si on n'a pas pu terminer la mesure,
            (portee->               // et cette mesure n'est pas la seule de
                                       la portée,
                  NombreMesures() > 0))
      {     partition->Ton          // on restaure le ton avant traitement
                                       des codes de la mesure
                  (sauvegardeTon);
            return TRUE;
      }
      melodie = codeATraiter;       // On avance dans la mélodie
      if (!                         // On ajoute les barres de notes dans
                                       leur brouillon
            AjouteSignetsBarresNotes()) return FALSE;
      return TRUE;
}

// Fonction de translation verticale de tous les objets graphiques
// de la mesure (on en profite pour initialisé le pointeur des notes
// graphiques vers leur portée
void CDecoupageMesure::TranslationVerticale(long decalage)
{     CVecteurIterateur             // Itérateur sur les objets graphiques
                                       de la mesure
            prochainObjet(graphiques);
      CGraphiqueMusical *           // Objet à translater
            objetGraphique;
      while (objetGraphique =       // En itérant sur les objets du vecteur,
            (CGraphiqueMusical *) prochainObjet())
      {     objetGraphique->        // on translate l'ordonnée des objets
                  PositionY(objetGraphique->PositionY() + decalage);
            if (objetGraphique->    // Si le graphique est un note,
                  Note())
                  ((CGraphiqueNote  // on met à jour la portée graphique de
                                       cette note
                        *) objetGraphique)->SaPortee(portee->SaPortee());
      }
}

// Réorganisation des graphiques en fonction des pixels accordés et du
// décalage subi
void CDecoupageMesure::ArrangeGraphiques
      (short decalage, short pixelsSupplementaires)
{     CVecteurIterateur             // Itérateur sur les objets graphiques
                                       de la mesure
            prochainGraphique(graphiques);
      CGraphiqueMusical *           // Objet graphique traité
            objetGraphique;
      long pixelsAjoutes = 0L,      // Nombre de pixels déjà ajoutés à la
                                       mesure
           pixelsIdeauxConsommes    // Nombre des décalages idéaux subis
                  = 0L,
           totalPixelsIdeaux = 0L,  // Somme des décalages ideaux de la
                                       mesure
           decalageObjetCourant;    // Nombre de pixels de décalage à faire
```

```
                                    subir par les objets suivants
      if (!((CGraphiqueMusical *)   // Si la mesure est incomplète, on a
                                    terminé
            graphiques.last())->GraphiqueFinMesure())
      {     while (objetGraphique = // En itérant sur les graphiques de la
                                    mesure,
                  (CGraphiqueMusical *) prochainGraphique())
            {     objetGraphique->  // On décale le graphique courant
                        PositionX(objetGraphique->PositionX() + decalage);
            }
            return;
      }
      while (objetGraphique =       // En itérant sur les graphiques de la
                                    mesure,
            (CGraphiqueMusical *) prochainGraphique())
      {     totalPixelsIdeaux +=    // On somme les nombres de pixels idéaux
                                    de chaque graphique
                  objetGraphique->DecalageIdeal();
      }
      prochainGraphique.reset();    // On repositionne l'itérateur au début
                                    du vecteur
      while (objetGraphique =       // En itérant sur les graphiques de la
                                    mesure,
            (CGraphiqueMusical *) prochainGraphique())
      {     pixelsIdeauxConsommes +=// On incrémente le nombre de pixels
                                    idéaux consommés
                  objetGraphique->DecalageIdeal();
            decalageObjetCourant =  // On calcule le décalage à faire subir
                                    aux objets suivants l'objet courant
                  totalPixelsIdeaux == 0?
                  0:                // Cas extrême où aucun des graphiques
                                    de la mesure ne peut être chewingommé
                  (pixelsIdeauxConsommes * pixelsSupplementaires)
                   / totalPixelsIdeaux
                  - pixelsAjoutes;
            objetGraphique->        // On décale le graphique courant
                  PositionX(objetGraphique->PositionX() + pixelsAjoutes +
                        decalage);
            pixelsAjoutes +=        // On incrémente le nombre de pixels
                                    ajouté et le décalage à faire subir
                                    aux prochains graphiques
                  decalageObjetCourant;
      }
}

// Réinitialisation du vecteur des objets graphiques de la mesure.
// ATTENTION, cette fonction ne fonctionne que si la mesure est la dernière
// insérée dans le vecteur des graphiques musicaux !
void CDecoupageMesure::EffaceObjetsGraphiques()
{     CVecteur *objetsGraphiques    // Liste des objets graphiques de la
                                    partition
            = partition->ObjetsGraphiques();
      CVecteur *liaisons            // Liste des liaisons de la partition
            = partition->BrouillonLiaisons();
      int nombreGraphiquesMesure =  // Nombre de graphiques de la mesure à
                                    effacer
            graphiques.entries();
      while                         // On efface chaque objet de la mesure
                                    de la partition
            (nombreGraphiquesMesure--)
            delete objetsGraphiques->pop();
      graphiques.clear();           // On n'a plus d'objets dans la mesure
      if (liaisons->entries())      // On supprime les liaisons ajoutées
                                    pendant la mesure
```

```
	{	while (liaison->entries() && ((CBrouillonGraphiqueLiaison *)
			liaisons->last())->IndiceDebut()
				> objetsGraphiques->entries())
		{	liaisons->pop();
		}
	}
}

// Ajout du graphique de continuateur à la fin de cette mesure
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::AjouteContinuateur()
{	CVecteur *objetsGraphiques	// Liste des objets graphiques de la
					   partition
		= partition->ObjetsGraphiques();
	CGraphiqueMusical *		// Objet graphique du continuateur de
					   mesure
		continuateur;
	float echelle = partition->	// Echelle de redimensionnement des
					   graphiques musicaux
		Echelle();
	if (!(continuateur = new	// On crée le nouvel objet graphique
		CGraphiqueContinuateurDeMesure(positionX + largeur + 2,
			long(19 * echelle)))) return FALSE;
	objetsGraphiques->push		// On le note dans la partition
		(continuateur);
	graphiques.push(continuateur);// On le note dans la mesure
	largeur += 2 +			// On met à jour de la largeur de la
					   mesure
		int(23 * echelle);
	return TRUE;
}

// Ajout des graphiques de la clé et du ton courant
// Renvoie FALSE si cela n'a pas été possible
BOOLEAN CDecoupageMesure::AjouteArmature
	(const unsigned char *&codeATraiter)
{	short ancienTon = partition	// Sauvegarde du ton précédent les codes
					   traités ci_dessous
		->Ton();
	while((*codeATraiter & 0xF0u) // Si on a un code de clé ou de ton
		== 0x90u)
	{	if (*codeATraiter	// On traite le changement de ton
			& 0x0Fu)
			TraiteCodeCle(codeATraiter);
		else			// On traite le changement de clé
			TraiteCodeTon(codeATraiter);
		codeATraiter +=3;	// On se positionne sur le code suivant
	}
	if (!AjouteCle())		// On ajoute la clé courante à la mesure
		return FALSE;
	AjouteTonalite();
	return TRUE;
}

// Ajout des graphiques des barres de notes (dans la partition)
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::AjouteBarresNotes()
{	float echelle = partition->	// Echelle de redimensionnement des
					   graphiques musicaux
		Echelle();
	CVecteurIterateur		// Itérateur sur les barres de notes
		barreSuivante(brouillonBarresNotes);
	CBrouillonGraphiqueBarre *barre;
	long ordonneeSi = portee->	// Ordonnée de la troisième ligne de la
```

```
                                    portée graphique courante
            SaPortee()->PositionTroisiemeLigne(echelle);
      while (barre =                // On demande à chaque brouillon de
                                    barre de la mesures
            (CBrouillonGraphiqueBarre *) barreSuivante())
      {     barre->AjusteHampe      // de s'occuper des hampes de ses notes
                  (&graphiques, ordonneeSi, echelle);
            ChangeDepassements      // On change les dépassements haut et
                                    bas de la mesure si c'est nécessaire
                  (barre->DepassementHaut(), barre->DepassementBas());
            if (!barre->            // de s'ajouter leurs graphiques de
                                    barre dans la mesure
                  MiseAuPropre(this, NULL)) return FALSE;
      }
      return TRUE;
}

// Ajout des graphiques des n-olets (dans la partition)
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CDecoupageMesure::AjouteNOlets()
{     float echelle = partition->   // Echelle de redimensionnement des
                                    graphiques musicaux
            Echelle();
      CVecteurIterateur nOletSuivant// Itérateur sur les n-olets
            (brouillonNOlets);
      CBrouillonGraphiqueNOlet *nOlet;
      long ordonneeSi = portee->    // Ordonnée de la troisième ligne de la
                                    portée graphique courante
            SaPortee()->PositionTroisiemeLigne(echelle);
      while (nOlet =                // On demande à chaque brouillon de
                                    barre de la mesures
            (CBrouillonGraphiqueNOlet *) nOletSuivant())
      {     if (!nOlet->AjusteHampe // de s'occuper des hampes de ses notes
                                    (juste les repositionner au besoin)
                  (&graphiques, echelle, ordonneeSi)) return FALSE;
            if (!nOlet->            // d'ajouter leurs graphiques de n-olet
                                    dans la mesure
                  MiseAuPropre(this))
                  return FALSE;
            ChangeDepassements      // On change les dépassements haut et
                                    bas si c'est nécessaire
                  (nOlet->DepassementHaut(), nOlet->DepassementBas());
      }
      return TRUE;
}

#endif
```

Annexe 6 : CBrouillonGraphiqueHorizontal

Cette annexe contient le listing des fichiers CBroHori.cpp et CBroHori.h.

CBroHori.h

```
// Nature: Définition de la classe décrivant un brouillon d'un graphique
// horizontal (barre de note, n-olet ou liaison)
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 14/11/96
// Modifications:
// Commentaires:

#ifndef CBrouillonGraphiqueHorizontal_H
#define CBrouillonGraphiqueHorizontal_H

#include "xvt.h"

#include "WaveDef.h"
#include CObjet_i
#include CVecteur_i

class CGraphiqueMusical;
class CVecteurIterateurGraphiques;
class CDecoupageMesure;
class CDecoupagePortee;

class CGraphiqueNote;

class CBrouillonGraphiqueHorizontal : public CObjet
{     protected:                    // Implémentation en mémoire:
            short indiceDebut;      //    Indice du début des graphiques
                                          reliés par ce graphique horizontal
            short nombreNotes;      //    Nombre de notes incluses dans ce
                                          graphique
            int depassementHaut,    //    Dépassement du graphique par
                                          rapport au point supérieur gauche
                                          de la portée
                depassementBas;
            CGraphiqueNote* premiereNote,
                                    //    Première note du graphique
                                          horizontal
                        *derniereNote;
                                    //    Dernière note du graphique
                                          horizontal

                                    // Fonctions à usage protégé:
            CGraphiqueNote *        //    Recherche de la nième note à
                                          partir d'un indice dans un vecteur
                                          de graphique musicaux
                  ChercheNiemeNote(const CVecteur *, int, int) const;

      public:                       // Interface standard:
            CBrouillonGraphiqueHorizontal
                                    //    Constructeur
                  (short, short);
            int IndicePremiereNote  //    Accès à l'indice de la première
                                          note du graphique horizontal
                  (CVecteur *) const;
            int IndiceDerniereNote  //    Accès à l'indice de la dernière
                                          note du graphique horizontal
                  (CVecteur *) const;
```

```
            inline short            //    Accès à l'indice du début de
                                          graphique
                  IndiceDebut() const {return indiceDebut;}
            inline short            //    Accès à l'indice de la première
                                          note
                  NombreNotes() const {return nombreNotes;}
            inline int              //    Accès au dépassement haut
                  DepassementHaut() const {return depassementHaut;}
            inline int              //    Accès au dépassement bas
                  DepassementBas() const {return depassementBas;}
};

class CBrouillonGraphiqueLiaison : public CBrouillonGraphiqueHorizontal
{     private:                      // Implémentation en mémoire:
            CVecteur *graphiques;   //    Graphiques musicaux de la
                                          partition
            int ordonneeSommet;     //    Ordonnée du sommet de la liaison
            float echelle;          //    Echelle de redimentionnement des
                                          graphiques musicaux

                                    // Fonctions à usage protégé:
            void CalculeCoordonnees //    Calcul des coordonnées de départ
                                          et d'arrivée d'un arc de liaison
                  (PNT &, int, short, BOOLEAN) const;
            BOOLEAN                 //    Renvoie l'orientation d'un arc
                                          d'une liaison
                  DetermineOrientation(int, int, long) const;
            void CalculeCarre       //    Calcul d'un carré contenant le
                                          cercle incluant l'arc d'une
                                          liaison
                  (RCT &, const PNT *, const PNT *, int, int, BOOLEAN);

      public:                       // Interface standard:
            CBrouillonGraphiqueLiaison(short, short, CVecteur *, float);
                                    //    Constructeur
            BOOLEAN MiseAuPropre    //    Transformation du brouillon en
                                          graphique de liaison
                  (const CDecoupagePortee *, int, int);
};

class CBrouillonGraphiqueNOlet : public CBrouillonGraphiqueHorizontal
{     private:                      // Implémentation en mémoire:
            short n;                //    N du N-olet
            float coefficientRythmique;
            int sommeDistancesSi;   //    Somme des distances entre les
                                          notes et la troisième barre de la
                                          portée

      public:                       // Interface standard:
            CBrouillonGraphiqueNOlet//    Constructeur
                  (short, float, short, short);
            BOOLEAN AjusteHampe     //    Ajustement de la direction des
                                          hampes d'un groupe de notes d'un
                                          n-olet
                  (CVecteur *, float, long);
            BOOLEAN MiseAuPropre    //    Transformation du brouillon en
                                          graphique de n-olet
                  (CDecoupageMesure *);
            inline float            //    Calcul du coéfficient de réduction
                                          du rythme conséquent à ce n-olet
                 CoefficientRythmique() const
                 {return coefficientRythmique;}
};
```

```
class CBrouillonGraphiqueBarre : public CBrouillonGraphiqueHorizontal
{     private:                        // Implémentation en mémoire:
            short niveau;             //    Niveau dans la hiérarchie des
                                            barres
            CVecteur sousBarres;      //    Sous-barres contenues dans cette
                                            barre
            float                     //    Coéfficient angulaire de la barre
                                            (doit être commun aux sous-barres)
                  coefficientAngulaire;
            int nombreGraphiques;     //    Nombre de graphique musicaux
                                            inclus dans cette barre
            int sommeDistancesSi;     //    Somme des distances entre les
                                            notes et la troisième barre de la
                                            portée

      public:                         // Interface standard:
            CBrouillonGraphiqueBarre//    Constructeur
                  (short, short);
            short AjouteGraphiques    //    Ajout de graphiques dans la barre
                  (CDecoupageMesure *, int &, short &);
            void AjusteHampe          //    Changement de direction et de
                                            hauteur des hampes
                  (const CVecteur *, long, float);
            BOOLEAN MiseAuPropre      //    Ajout de la barre graphique dans
                                            la mesure
                  (CDecoupageMesure *, CBrouillonGraphiqueBarre *);
            inline CGraphiqueNote *
                  PremiereNote(CVecteur *graphiques) const
                  {return (CGraphiqueNote *)
                   graphiques->at(IndicePremiereNote(graphiques));}
            inline float              //    Accès au coéfficient angulaire de
                                            la barre
                  CoefficientAngulaire() const
                  {return coefficientAngulaire;}
            inline int                //    Accès au nombre de graphique
                                            musicaux inclus dans cette barre
                  NombreGraphiques() const {return nombreGraphiques;}
};

#endif
```

CBroHori.cpp

```
// Nature: Implémentation de la classe décrivant un brouillon d'objet
// graphique horizontal
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 14/11/96
// Modifications:
// Commentaires:
#ifdef MUSIQUE

#include "MusiDef.h"
#include CBrouillonGraphiqueHorizontal_i
#include CGraphiqueNotes_i
#include CGraphiqueSilences_i
#include CGraphiquePortee_i
#include CGraphiqueAutres_i
#include CDecoupageMesure_i
#include CDecoupagePortee_i
#include CDecoupagePartition_i

#include ConstantesMusique_i
```

```
#include "WaveDef.h"
#include CVecteurIterateur_i

#include "math.h"

// Constructeur
CBrouillonGraphiqueHorizontal::CBrouillonGraphiqueHorizontal
      (short debut, short nombre)
      : CObjet()
{     indiceDebut = debut;          // On note le début et le nombre de note
                                       du graphique horizontal
      nombreNotes = nombre;
      premiereNote = NULL;          // On ne connait pas encore ni la
                                       première note, ni la dernière
      dernieresNote = NULL;
      depassementHaut =             // On n'a pas encore de dépassement
            depassementBas = 0;
}

// Fonction privée de recherche de la nième note suivant un indice
// dans un vecteur de graphiques musicaux
CGraphiqueNote *CBrouillonGraphiqueHorizontal::ChercheNiemeNote
      (const CVecteur *vecteurGraphiques, int indice, int n) const
{     int nombreGraphiquesPartition // Nombre de graphiques de la partition
            = vecteurGraphiques->entries();
      for (;                        // Tant qu'on n'est pas arrivé à la fin
                                       du vecteur
           indice < nombreGraphiquesPartition; indice++)
      {     if ((                   // Si l'objet à cet indice dans le
                                       vecteur est un note,
               (CGraphiqueMusical *)vecteurGraphiques->at(indice))->Note())
            {     n--;              // On a traité une note
                  if (!n) return    // Si c'était la nième note, on la
                                       renvoie
                        (CGraphiqueNote *) vecteurGraphiques->at(indice);
            }
      }
      return NULL;
}

// Calcul de l'indice de la première note du graphique horizontal
int CBrouillonGraphiqueHorizontal::IndicePremiereNote(CVecteur
*vecteurGraphiques) const
{     int indice,                   // Indice courant
          nombreGraphiquesPartition // Nombre de graphiques de la partition
            = vecteurGraphiques->entries();
      for (indice = indiceDebut;    // Tant que l'on n'a pas trouvé la
                                       dernière note et qu'il reste des
                                       graphiques,
           indice < nombreGraphiquesPartition;
           indice++)
      {     if (((CGraphiqueMusical // On regarde si ce graphique est une
                                       note
                  *) vecteurGraphiques->at(indice))->Note())
                        return indice;
      }
      return 0;
}

// Calcul de l'indice de la dernière note du graphique horizontal
int CBrouillonGraphiqueHorizontal::IndiceDerniereNote
      (CVecteur *vecteurGraphiques) const
{     int indice,                   // Indice courant
          nombreGraphiquesPartition // Nombre de graphiques de la partition
```

```
            = vecteurGraphiques->entries();
      short notesTrouvee = 0;       // Compteur de notes
      for (indice = indiceDebut;
           notesTrouvee <           // Tant que l'on n'a pas trouvé la
                                       dernière note et qu'il reste des
                                       graphiques,
            nombreNotes && indice < nombreGraphiquesPartition;
           indice++)
      {     if (((CGraphiqueMusical // On regarde si ce graphique est une
                                       note
                 *) vecteurGraphiques->at(indice))->Note())
                      notesTrouvee++;
      }
      return indice - 1;
}

// Cas de la liaison
// Fonction privée de calcul des coordonnées d'une note extrèmale
// d'une liaison
// depassement == -1 s'il y a dépassement à gauche,
//             0 s'il n'y en a pas
//             1 s'il y a dépassement à droite
void CBrouillonGraphiqueLiaison::CalculeCoordonnees
      (PNT &point, int indice, short depassement,
       BOOLEAN orientationDessus) const
{     CGraphiqueNote *note          // Graphique de la note correspondant à
                                       l'indice
            = (CGraphiqueNote *) graphiques->at(indice);
      point.v = orientationDessus?  // Le point se situe 3 pixels sous ou
                                       sur le point extrémal de la note
            short(note->OrdonneeMinimale(echelle) - 3):
            short(note->OrdonneeMaximale(echelle) + 3);
      if (depassement == -1)        // S'il y a dépassement à gauche,
      {     point.h = short(        // Le point se situe 50 pixels
                                       redimensionnés à gauche de la note
                  note->PositionX() - short(50 * echelle));
            return;
      }
      if (depassement == 1)         // S'il y a dépassement à droite,
      {
            point.h = short(        // Le point se situe 50 pixels
                                       redimensionnés à droite de la note
                  note->PositionX() + short(50 * echelle));
            return;
      }
      point.h = short(note->        // Le point se situe au milieu de la
                                       note
            PositionX() + short((LARGEUR_NOIRE >> 1) * echelle));
}

// Fonction privée : renvoie l'orientation d'une liaison
BOOLEAN CBrouillonGraphiqueLiaison::DetermineOrientation
      (int indicePremiereNote, int indiceDerniereNote,
      long positionTroisiemeLigne) const
{     CGraphiqueNote *note;
      int indice;                   // Indice itérateur sur le vecteur
      long distanceMaximumBas,      // Distance maximum du bas d'une note à
                                       la troisième ligne de la portée
           distanceMaximumHaut;     // Distance maximum du haut d'une note à
                                       la troisième ligne de la portée
      note = (CGraphiqueNote *)     // Première note de la liaison
            graphiques->at(indicePremiereNote);
      if (note->NoteHampe())        // Si la première note possède une
                                       hampe,
```

```
            return !(                  // L'orientation de la liaison est
                                          l'inverse de l'orientation de la
                                          hampe
                  (CGraphiqueNoteHampe *)note)->HampeHaute();
      note = ((CGraphiqueNote *)       // Dernière note de la liaison
            graphiques->at(indiceDerniereNote));
      if (note->NoteHampe())           // Si la dernière note possède une
                                          hampe,
            return !(                  // L'orientation de la liaison est
                                          l'inverse de l'orientation de la
                                          hampe
                  (CGraphiqueNoteHampe *)note)->HampeHaute();
      indice = indicePremiereNote;     // Si les deux notes extrêmes ne sont
                                          pas pourvues de hampes,
      distanceMaximumHaut              // On initialise les distances
            = distanceMaximumBas = 0;
      while (indice <=                 // On parcourt le vecteur entre les deux
                                          indices extrêmes
            indiceDerniereNote)
      {     if (((CGraphiqueMusical    // Si le graphique courant est une note,
                  *) graphiques->at(indice))->Note())
            {     note =               // Note courante
                        (CGraphiqueNote *) graphiques->at(indice);
                  if (                 // On cherche la position de la note la
                                          plus haute
                        positionTroisiemeLigne -
                        note->OrdonneeMinimale(echelle)
                          > distanceMaximumHaut)
                              distanceMaximumHaut = positionTroisiemeLigne -
                              note->OrdonneeMinimale(echelle);
                  if (note->           // et la position de la note la plus
                                          basse
                        OrdonneeMaximale(echelle) - positionTroisiemeLigne
                         > distanceMaximumBas)
                              distanceMaximumBas =
                                 note->OrdonneeMaximale(echelle) -
                                 positionTroisiemeLigne;
            }
            indice++;                  // On incrémente l'indice
      }                                // Règle : Les notes extrêmes
                                          interviennent dans la décision
      return (distanceMaximumBas       // de l'orientation de la liaison.
            < distanceMaximumHaut);
}

// Fonction privée de calcul du rectangle contenant l'ellipse
// d'une liaison
void CBrouillonGraphiqueLiaison::CalculeCarre
      (RCT &carreContenant,
       const PNT *pointDebut, const PNT *pointFin,
       int indiceDebut, int indiceFin,
       BOOLEAN orientationDessus)
{     int indice;                      // Compteur dans le vecteur des
                                          graphiques
      CGraphiqueNote *note;            // Pointeur d'objet après cast
      long pointReference;             // Point de référence sur le graphique
                                          de la note pour le calcul du
                                          coefficient angulaire
      float coefficientPointDebut,     // Coefficient angulaire d'une certaine
                                          note au point de début et de fin de
                                          l'arc
            coefficientPointFin,
            coefficientCorde1,         // Coefficient angulaire de la corde
                                          liée au point de début de l'arc
```

```
            coefficientCorde2,      // Coefficient angulaire de la corde
                                       liée au point de fin de l'arc
            coefficientAB,          // Coefficient angulaire de la droite
                                       reliant les points de début et de fin
            coefficientMin;         // Coefficient angulaire de la droite
                                       reliant les extrémités à un point
                                       situé à 10 pixels du milieu de [AB]
      int longueurLiaison;          // Longueur de la liaison en pixels
      PNT intersectionCordes,       // Intersection des deux cordes
          milieuCorde1,             // Milieux des cordes
          milieuCorde2,
          centreCercle;             // Centre du cercle incluant l'arc
      int rayon;                    // Rayon du cercle
      indice = indiceDebut + 1;
      longueurLiaison =             // On calcule la longueur de la liaison
            pointFin->h - pointDebut->h;
      coefficientAB =               // On calcule le coefficient angulaire
                                       de la droite reliant point de début
                                       et de fin
            float(pointDebut->v - pointFin->v) /
               float(pointDebut->h - pointFin->h);
      coefficientMin =              // On calcule le coefficient minimum
            10.F / longueurLiaison;
      if (orientationDessus)        // On initialise les coefficients de
                                       corde extrêmes
      {     coefficientCorde1 =
//                tan(atan(coefficientAB) - atan(coefficientMin));
                  (coefficientAB - coefficientMin) /
                     (1.0F + coefficientAB * coefficientMin);

            coefficientCorde2 =
//                tan(atan(coefficientAB) + atan(coefficientMin));
                  (coefficientAB + coefficientMin) /
                     (1.0F - coefficientAB * coefficientMin);
      }
      else
      {     coefficientCorde1 =
//                tan(atan(coefficientAB) + atan(coefficientMin));
                  (coefficientAB + coefficientMin) /
                     (1.0F - coefficientAB * coefficientMin);
            coefficientCorde2 =
//                tan(atan(coefficientAB) - atan(coefficientMin));
                  (coefficientAB - coefficientMin) /
                     (1.0F + coefficientAB * coefficientMin);
      }
      while (indice < indiceFin)
      {     if (((CGraphiqueMusical // Si le graphique est une note,
                  *) graphiques->at(indice))->Note())
            {     note =            // On note cette note
                        (CGraphiqueNote *) graphiques->at(indice);
                  pointReference =  // On calcule le point de référence
                        orientationDessus?
                        note->OrdonneeMinimale(echelle) -
                           short(10 * echelle):
                        note->OrdonneeMaximale(echelle) +
                           short(10 * echelle);
                  coefficientPointDebut =
                                    // On calcule le coefficient angulaire
                                       par rapport au point de début de la
                                       liaison
                        float(pointDebut->v - pointReference) /
                           float(pointDebut->h - note->PositionX());
                  coefficientPointFin =
                                    // On calcule le coefficient angulaire
```

```
                                    par rapport au point de début de la
                                    liaison
                        float(pointFin->v - pointReference) /
                           float(pointFin->h - note->PositionX());
                  coefficientCorde1 // On met à jour si nécessaire le
                                    coefficient angulaire de la corde
                                    liée au point de début
                        = orientationDessus?
                        (coefficientCorde1 > coefficientPointDebut?
                         coefficientPointDebut: coefficientCorde1):
                         (coefficientCorde1 < coefficientPointDebut?
                          coefficientPointDebut: coefficientCorde1);
                  coefficientCorde2 // On met à jour si nécessaire le
                                    coefficient angulaire de la corde
                                    liée au point de fi,
                        = orientationDessus?
                        (coefficientCorde2 < coefficientPointFin?
                         coefficientPointFin: coefficientCorde2):
                         (coefficientCorde2 > coefficientPointFin?
                          coefficientPointFin: coefficientCorde2);
            }
            indice++;
      }
      intersectionCordes.h =        // On calcule l'intersection des deux
                                    cordes
            short((coefficientCorde1 * pointDebut->h -
                   coefficientCorde2 * pointFin->h -
                   pointDebut->v + pointFin->v)
                 / (coefficientCorde1 - coefficientCorde2));
      intersectionCordes.v =
            short(coefficientCorde1 * (intersectionCordes.h -
                  pointDebut->h) + pointDebut->v);
      milieuCorde1.h =              // On calcule les milieux des cordes
            (pointDebut->h + intersectionCordes.h) >> 1;
      milieuCorde1.v =
            (pointDebut->v + intersectionCordes.v) >> 1;
      milieuCorde2.h =
            (pointFin->h + intersectionCordes.h) >> 1;
      milieuCorde2.v =
            (pointFin->v + intersectionCordes.v) >> 1;
      if (!coefficientCorde1)       // On calcule le centre du cercle
                                    (intersection des deux médiatrices)
      {     centreCercle.h = milieuCorde1.h;
            centreCercle.v = short((milieuCorde2.h - centreCercle.h) /
               coefficientCorde2 + milieuCorde2.v);
      }
      else if (!coefficientCorde2)
      {     centreCercle.h = milieuCorde2.h;
            centreCercle.v = short((milieuCorde1.h - centreCercle.h) /
               coefficientCorde1 + milieuCorde1.v);
      }
      else
      {     centreCercle.h =
                  short(((coefficientCorde1 * coefficientCorde2)
                         / (coefficientCorde1 - coefficientCorde2))
                        * (milieuCorde2.h / coefficientCorde2
                          - milieuCorde1.h / coefficientCorde1
                          + milieuCorde2.v - milieuCorde1.v));
            centreCercle.v =
                  short((milieuCorde1.h - centreCercle.h) /
                     coefficientCorde1 + milieuCorde1.v);
      }
      rayon = int(sqrt              // On calcule le rayon du cercle
            (  long(centreCercle.h - pointDebut->h) *
```

```
                  long(centreCercle.h - pointDebut->h)
                + long(centreCercle.v - pointDebut->v) *
                  long(centreCercle.v - pointDebut->v)));
      if (rayon > SHRT_MAX)          // S'il y a dépassement de capacité,
      {     xvt_rect_set             // on renvoie un carré vide
                  (&carreContenant, 0, 0, 0, 0);
            return;
      }
      ordonneeSommet = orientationDessus?
            centreCercle.v - rayon: centreCercle.v + rayon;
      xvt_rect_set(&carreContenant, // On calcule les coordonnées du carré
                                       contenant le cercle
            centreCercle.h - rayon, centreCercle.v - rayon,
            centreCercle.h + rayon, centreCercle.v + rayon);
}

// Constructeur
CBrouillonGraphiqueLiaison::CBrouillonGraphiqueLiaison
      (short debut, short nombre, CVecteur *graphiquesMusicaux,
       float echelleDesiree)
      : CBrouillonGraphiqueHorizontal(debut, nombre)
{     graphiques = graphiquesMusicaux;
      echelle = echelleDesiree;
      ordonneeSommet = 0;
}

// Création d'un arc de la liaison (qui peut être composée de
// plusieurs arcs)
BOOLEAN CBrouillonGraphiqueLiaison::MiseAuPropre
      (const CDecoupagePortee *portee, int indiceDebut, int indiceFin)
{     BOOLEAN orientationDessus;    // Orientation de la liaison (au-dessus
                                       ou en-dessous des notes)
      PNT coordonneesDebut,         // Coordonnée du point de début et de
                                       fin de la liaison
          coordonneesFin;
      RCT carre;                    // Carré contenant le cercle qui inclut
                                       la liaison
      int indice1iereNote,          // Indice de la première et de la
                                       dernière note incluses dans l'arc de
                                       la liaison
          indiceDerniereNote;
      long abscisse,                // Abscisse et ordonnée du graphique de
                                       la liaison
           ordonnee;
      CGraphiqueLiaison *liaison;   // Liaison à insérer dans les vecteurs
                                       de graphiques musicaux
      CVecteur *graphiquesPortee =  // Objets de la première mesure de la
                                       portée
            portee->PremiereMesure()->Graphiques();
      long positionTroisiemeLigne   // Position de la ligne du milieu du
                                       graphique de la portée
            = portee->SaPortee()->
                  PositionTroisiemeLigne(portee->Echelle());
      indice1iereNote = indiceDebut == -1?
                        portee->PremiereMesure()->IndicePremiereNote():
                        indiceDebut;
      indiceDerniereNote = indiceFin == -1?
                        portee->DerniereMesure()->IndiceDerniereNote():
                        indiceFin;
      orientationDessus =           // On détermine l'orientation de la
                                       liaison
            DetermineOrientation
                  (IndicePremiereNote(graphiques),
                   IndiceDerniereNote(graphiques), positionTroisiemeLigne);
```

```
      CalculeCoordonnees            // On calcule les coordonnées du point
                                       de début de la liaison
            (coordonneesDebut, indice1iereNote,
             indiceDebut == -1? -1: 0, orientationDessus);
      CalculeCoordonnees            // On calcule les coordonnées du point
                                       de fin de la liaison
            (coordonneesFin, indiceDerniereNote,
             indiceFin == -1? 1: 0, orientationDessus);
      CalculeCarre(carre,           // On calcule le carré contenant le
                                       cercle contenant l'arc de la liaison
            &coordonneesDebut, &coordonneesFin,
            indice1iereNote, indiceDerniereNote, orientationDessus);
      abscisse = carre.left;        // On en déduit les coordonnées de
                                       l'objet graphique
      ordonnee = carre.top;
      if (!(liaison = new           // On crée le graphique de la liaison
            CGraphiqueLiaison(abscisse, ordonnee, &carre,
             orientationDessus? &coordonneesFin: &coordonneesDebut,
             orientationDessus? &coordonneesDebut: &coordonneesFin)))
            return FALSE;
      graphiques->push(liaison);
      graphiquesPortee->push(liaison);
      depassementHaut =             // On initialise les dépassements
            -int(ordonneeSommet / echelle);
      depassementBas = int(ordonneeSommet / echelle) -
            (HAUTEUR_INTERLIGNE << 2);
      return TRUE;
}

// Cas du n_olet
// Constructeur
CBrouillonGraphiqueNOlet::CBrouillonGraphiqueNOlet
      (short nDesire, float coefficient, short debut, short nombre)
      : CBrouillonGraphiqueHorizontal(debut, nombre)
{     n = nDesire;
      coefficientRythmique = coefficient;
      sommeDistancesSi = 0;
}

// Ajustement de la position des hampes d'un groupe de notes d'un n-olet
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CBrouillonGraphiqueNOlet::AjusteHampe
      (CVecteur *graphiques, float echelle, long ordonneeSi)
{     CGraphiqueNote *niemeNote;    // n ième note du n-olet
      CGraphiqueNoteHampe *         // n ième note lorsqu'elle possède une
                                       hampe
            noteHampe;
      short numeroNote;             // Compteur des notes de la barre
      numeroNote = 1;               // On commence sur la première note de
                                       la barre
      while (numeroNote <=          // On calcule la somme des distances des
                                       notes à la ligne du milieu de la
                                       portée
            nombreNotes)
      {     niemeNote =             // On se positionne sur la note courante
                  (CGraphiqueNote *)
                     ChercheNiemeNote(graphiques, indiceDebut, numeroNote);
            if (!niemeNote)         // Problème si le groupe est coupé
                                       (mesure trop petite)
                  return FALSE;
            sommeDistancesSi += int(niemeNote->PositionY() - ordonneeSi);
            numeroNote++;           // On passe à la note suivante
      }
      numeroNote = 0;
```

```
      while (numeroNote++ < nombreNotes)
      {     niemeNote =             // On se positionne sur la note suivante
                  (CGraphiqueNote *) ChercheNiemeNote
                  (graphiques, indiceDebut, numeroNote);
            if (!niemeNote) return FALSE;
            if (niemeNote->         // Si on a affaire à une note avec
                                       hampe,
                  NoteHampe())
            {     noteHampe = (CGraphiqueNoteHampe *) niemeNote;
                  noteHampe->       // on positionne la hampe correctement
                                       selon le groupe de note
                        HampeHaute(sommeDistancesSi < 0? FALSE: TRUE);
            }
      }
      return TRUE;
}

// Transformation du brouillon de n-olet en un n-olet graphique
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CBrouillonGraphiqueNOlet::MiseAuPropre(CDecoupageMesure *mesure)
{     float echelle = mesure->      // Echelle de redimensionnement des
                                       graphiques musicaux
            Partition()->Echelle();
      CVecteur *graphiques          // Vecteur des graphiques de la mesure
            = mesure->Graphiques();
      long ordonneeSi = mesure->    // Ordonnée de la troisième ligne de la
                                       portée graphique courante
            Portee()->SaPortee()->PositionTroisiemeLigne(echelle);
      short positionX,              // Coordonnée de l'objet graphique par
                                       rapport au point supérieur gauche de
                                       la portée
            positionY,
            ordonneePointDebut,     // Ordonnée des points de début et de
                                       fin
            ordonneePointFin,
            ordonneePointLigne,     // Ordonnée d'un point de la ligne
                                       diagonale du n-olet
            ordonneeNote,           // Ordonnée d'une note du n-olet
            numeroNote;             // Compteur de note dans le n-olet
      CGraphiqueNote *niemeNote;    // n ième note du n-olet
      float coefficientAngulaire;   // coefficient angulaire de la droite
                                       reliant le point de début au point de
                                       fin
      PNT pointDebut,               // Points de début et de fin du n-olet
                                       (en coordonnées relative à l'objet
                                       graphique)
          pointFin;
      CGraphiqueNolet *nolet;       // Graphique du n-olet à insérer
      if (!premiereNote)            // Si les notes extrémales ne sont pas
                                       encore initialisées,
      {     premiereNote =          // On retrouve la première note du n-
                                       olet
                  (CGraphiqueNote *) ChercheNiemeNote
                        (graphiques, indiceDebut, 1);
            derniereNote =          // On retrouve la dernière note du n-
                                       olet
                  (CGraphiqueNote *) ChercheNiemeNote
                        (graphiques, indiceDebut, nombreNotes);
      }
      ordonneePointDebut =          // On calcule l'ordonnée du point de
                                       début du graphique du n-olet
            sommeDistancesSi >= 0?
            short(premiereNote->    // Cas des notes basses (sur la portée)
                  PositionY() + short(40 * echelle)):
```

```
      short(premiereNote->    // Cas des notes hautes (sur la portée)
            PositionY() - short(20 * echelle));
ordonneePointFin =            // On calcule l'ordonnée du point de fin
                                 du graphique du n-olet
      sommeDistancesSi >= 0?
      short(derniereNote->    // Cas des notes basses (sur la portée)
            PositionY() + short(40 * echelle)):
      short(derniereNote->    // Cas des notes hautes (sur la portée)
            PositionY() - short(20 * echelle));
pointDebut.h = 0;             // On calcule les coordonnées du point
                                 de début de la barre
pointDebut.v =                // par rapport au point supérieur gauche
                                 du graphique
      ordonneePointFin < ordonneePointDebut?
      short(ordonneePointDebut - ordonneePointFin): 0;
pointFin.h = short(           // On calcule les coordonnées du point
                                 de fin de la barre
      derniereNote->          // par rapport au point supérieur gauche
                                 du graphique
      PositionX() - premiereNote->PositionX());
pointFin.v = ordonneePointFin < ordonneePointDebut?
      0: short(ordonneePointFin - ordonneePointDebut);
positionX = short(            // On calcule les coordonnées du
                                 graphique
      premiereNote->          // par rapport au point supérieur gauche
                                 de la partition
      PositionX() + short(LARGEUR_NOIRE / 2 * echelle));
positionY = pointFin.v?
            ordonneePointDebut:
            ordonneePointFin;
coefficientAngulaire =        // On calcule le coefficient angulaire
                                 de la droite reliant les points de
                                 début et de fin
      float(pointDebut.v - pointFin.v) /
      float(pointDebut.h - pointFin.h);
numeroNote = 0;               // On va décaler verticalement si le
                                 graphique du n-olet collisionne une
                                 note
while (numeroNote++ < nombreNotes)
{     niemeNote =             // On se positionne sur la note suivante
            (CGraphiqueNote *) ChercheNiemeNote
                  (graphiques, indiceDebut, numeroNote);
      if (!niemeNote) return FALSE;
      ordonneePointLigne =    // On calcule l'ordonnée du point de la
                                 ligne diagonale du n-olet dont
                                 l'abscisse et celle de la nième note
            (positionY + pointDebut.v)
            + short(coefficientAngulaire
                  * (niemeNote->PositionX()
                     - (positionX + pointDebut.h)
                    )
                  );
      ordonneeNote =          // On calcule l'ordonnée extrêmale de la
                                 note
            sommeDistancesSi  // si le n-olet est sous les notes
            >= 0? short(niemeNote->OrdonneeMaximale(echelle)
                  + 20 * echelle):
                  short(niemeNote->OrdonneeMinimale(echelle)
                  - 20 * echelle);
      if ((sommeDistancesSi   // Si la note dépasse soit au-dessus,
                                 soit au-dessous de la barre
            >= 0) == (ordonneeNote > ordonneePointLigne))
      {     positionY +=      // On baisse ou surélève la barre
                  ordonneeNote - ordonneePointLigne;
```

```
            }
      }
      depassementHaut =             // On met à jour les dépassements
                                    // 35 est la hauteur approximative d'un
                                       graphique de n-olet (avec le chiffre)
            pointDebut.v > pointFin.v?
            -int(pointFin.v / echelle) - 35:
            -int(pointDebut.v / echelle) - 35;
      depassementBas =
            pointDebut.v > pointFin.v?
            int(pointDebut.v / echelle) + 35:
            int(pointFin.v / echelle) + 35;
      if (!(nolet = new             // On crée la barre de note graphique
            CGraphiqueNolet(positionX, positionY, &pointDebut, &pointFin,
                  n, sommeDistancesSi < 0))) return FALSE;
      graphiques->push(nolet);      // On la note dans la mesure
      mesure->Partition()->         // et dans la partition
            ObjetsGraphiques()->push(nolet);
      return TRUE;
}

// Cas de la barre de notes
// Constructeur
CBrouillonGraphiqueBarre::CBrouillonGraphiqueBarre
      (short debut, short niveauDesire)
      : CBrouillonGraphiqueHorizontal(debut, 0)
{     niveau = niveauDesire;        // On initialise les différents
                                       paramètres de l'objet
      nombreGraphiques = 0;
      sommeDistancesSi = 0;         // Les distances des notes au point de
                                       milieu de portée ne peut pas encore
                                       être calculées
}

// Ajout de graphiques dans la barre
// Pré: La note courante doit être un CGraphiqueNote
// Post: indiceGraphiqueCourant est l'indice de la dernière note de la //
// barre
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CBrouillonGraphiqueBarre::AjouteGraphiques
      (CDecoupageMesure *mesure, int &indiceGraphiqueCourant,
       short &tempsRestant)
{     CBrouillonGraphiqueBarre      // Sous-barre contenue dans cette barre
            *barre;
      int indiceDerniereNote        // Indice de la dernière note de la
                                       barre
            = indiceGraphiqueCourant;
      CVecteur *graphiques =        // Vecteur des graphiques de la mesure
            mesure->Graphiques();
      CGraphiqueNote *noteCourante  // Graphique courant lorsqu'il s'agit
                                       d'une note
            = (CGraphiqueNote *) graphiques->at(indiceGraphiqueCourant);
                                    // Invariant de la boucle:
                                    // * le temps accordé n'est pas épuisé,
                                    // * la noteCourante est une note
                                    // * indiceGraphiqueCourant est l'indice
                                         de la dernière note de la barre
                                    // * la barre commence nombreNotes avant
                                         la note courante
      while (tempsRestant > 0 &&    // * et se termine sur la note dont
                                         l'indice est indiceGraphiqueCourant
            noteCourante->NombreMoustaches() >= niveau)
      {     if (tempsRestant <      // On a terminé si on n'a plus de temps
                                       pour insérer la note courante
```

```
      noteCourante->Duree())
{     indiceGraphiqueCourant = indiceDerniereNote;
                        // Le graphique courant est la dernière
                           note de la barre
      return TRUE;
}
if (noteCourante->      // Si cette note se situe au niveau de
                           la barre,
      NombreMoustaches() == niveau)
{     tempsRestant -=   // On diminue le temps restant dans la
                           barre
            noteCourante->Duree();
      nombreNotes++;    // On ajoute cette note dans la barre
}
else                    // Si la note se situe dans un niveau de
                           barre inférieur à la barre courante
{     if (!(barre = new // On crée une nouvelle barre de niveau
                           supérieur
            CBrouillonGraphiqueBarre
                  (indiceGraphiqueCourant, niveau + 1)))
            return FALSE;
      sousBarres.push   // On note la sous-barre
            (barre);
      if (!barre->      // On ajoute des graphiques à cette
                           barre
            AjouteGraphiques(mesure,
                  indiceGraphiqueCourant, tempsRestant))
            return      // On se trouve sur la dernière note de
                           la sous-barre:
                  FALSE;
      nombreNotes +=    // On ajoute le nombre de notes de la
                           sous-barre à la barre courante
            barre->NombreNotes();
}
indiceDerniereNote =    // On enregistre l'indice de la dernière
                           note de la barre
      indiceGraphiqueCourant;
indiceGraphiqueCourant++;
                        // On passe au graphique suivant
if (                    // Si on a atteint la fin de la mesure,
                           on a terminé
      indiceGraphiqueCourant == graphiques->entries())
{     indiceGraphiqueCourant = indiceDerniereNote;
      return TRUE;      // et on n'oublie pas de renvoyer
                           l'indice de la dernière note de la
                           barre
}
while (!(               // On se positionne sur la note suivante
      (CGraphiqueMusical *)graphiques->
            at(indiceGraphiqueCourant))->Note())
{     if (              // Si le graphique courant est un
                           silence,
            ((CGraphiqueMusical *) graphiques->
                  at(indiceGraphiqueCourant))->Silence())
            tempsRestant// On diminue le temps restant dans la
                           barre
                  -= ((CGraphiqueSilence *) graphiques->
                  at(indiceGraphiqueCourant))->Duree();
                        // Si on a dépassé la fin de la mesure,
                           on a terminé
      if (++indiceGraphiqueCourant == graphiques->entries())
      {     indiceGraphiqueCourant = indiceDerniereNote;
            return TRUE;// et on n'oublie pas de renvoyer
                           l'indice de la dernière note de la
```

```
                                        barre
                    }
               }
               noteCourante =           // On est sûr que le graphique est une
                                           note
                    (CGraphiqueNote *) graphiques->
                         at(indiceGraphiqueCourant);
          }
          indiceGraphiqueCourant =      // On restaure l'indice de la dernière
                                           note de la barre
               indiceDerniereNote;
          return TRUE;
}

// Changement de direction et de hauteur des hampes de notes si une barre
// les relie
void CBrouillonGraphiqueBarre::AjusteHampe
          (const CVecteur *graphiques, long ordonneeSi, float echelle)
{         CGraphiqueNoteMoustachue      // Première et dernière note de la barre
               *niemeNote;              // n ième note de la barre
          long abscissePremiereNote,    // Coordonnée de la première note de la
                                           barre
               ordonneePremiereNote,
               depassementHautNote,     // Dépassements d'une note de la barre
               depassementBasNote;
          short numeroNote,             // Compteur des notes de la barre
               distance,                // Distance entre une note et la barre
               distanceMaximum,         // Distance maximum entre une note et la
                                           barre
               distanceMinimum;
          if (!premiereNote)            // Si les notes extrémales ne sont pas
                                           encore initialisées,
          {    premiereNote =           // On retrouve la première note du n-
                                           olet
                    (CGraphiqueNote *) ChercheNiemeNote(graphiques,
                         indiceDebut, 1);
               dernierеNote =           // On retrouve la dernière note du n-
                                           olet
                    (CGraphiqueNote *) ChercheNiemeNote(graphiques,
                         indiceDebut, nombreNotes);
          }
          abscissePremiereNote          // On calcule les coordonnées de la
                                           première note
               = premiereNote->PositionX();
          ordonneePremiereNote = premiereNote->PositionY();
          coefficientAngulaire =        // On calcule le coefficient angulaire
                                           de la barre
               float(ordonneePremiereNote - dernierеNote->PositionY())
               / float(abscissePremiereNote - dernierеNote->PositionX());
          numeroNote = 1;               // On commence sur la première note de
                                           la barre
          distanceMaximum =             // et les distances maxima sont nulles
               distanceMinimum = 0;
          while (numeroNote <           // On calcule la distance des notes à la
                                           droite reliant les deux notes
                                           extrêmes
                    nombreNotes)
          {    niemeNote =              // On se positionne sur la note courante
                    (CGraphiqueNoteMoustachue *) ChercheNiemeNote(graphiques,
                         indiceDebut, numeroNote);
               sommeDistancesSi += int(niemeNote->PositionY() - ordonneeSi);
               distance =               // On calcule la distance pour cette
                                           note
                    short(niemeNote->PositionY() -
```

```
                (ordonneePremiereNote +
                 long(coefficientAngulaire * (niemeNote->PositionX() -
                      abscissePremiereNote))));
          distanceMaximum =       // On met à jour les distances maximales
                distanceMaximum < distance +
                (niemeNote->NombreMoustaches() * HAUTEUR_INTERBARRE)?
                distance + ((niemeNote->NombreMoustaches() *
                HAUTEUR_INTERBARRE)): distanceMaximum;
          distanceMinimum =
                distanceMinimum > distance -
                (niemeNote->NombreMoustaches() * HAUTEUR_INTERBARRE)?
                distance - ((niemeNote->NombreMoustaches() *
                HAUTEUR_INTERBARRE)): distanceMinimum;
          numeroNote++;           // On passe à la note suivante
     }
     numeroNote = 0;
     while (numeroNote++ < nombreNotes)
     {    niemeNote =             // On se positionne sur la note suivante
                (CGraphiqueNoteMoustachue *) ChercheNiemeNote(graphiques,
                      indiceDebut, numeroNote);
          distance =              // On calcule la distance pour cette
                                     note
                short(niemeNote->PositionY() -
                (ordonneePremiereNote +
                 long(coefficientAngulaire * (niemeNote->PositionX() -
                      abscissePremiereNote))));
          niemeNote->HampeHaute(  // On positionne la hampe en haut
                sommeDistancesSi < 0? FALSE: TRUE);
          niemeNote->HauteurHampe // On calcule la hauteur souhaitée de
                                     la hampe
                (sommeDistancesSi < 0?
                      distanceMaximum + short(HAUTEUR_HAMPE_DEFAUT *
                            echelle) - distance:
                      -distanceMinimum + short(HAUTEUR_HAMPE_DEFAUT *
                            echelle) + distance);
          depassementHautNote =   // On note les dépassements de cette
                                     note
                sommeDistancesSi  // Si la hampe est basse, on ajoute
                                     l'épaisseur d'une barre au dépassemnt
                < 0?int(-niemeNote->OrdonneeMinimale(echelle) / echelle):
                    int(-niemeNote->OrdonneeMinimale(echelle) / echelle)
                      + EPAISSEUR_BARRE_NOTE;
          depassementBasNote =
                sommeDistancesSi  // Si la hampe est haute, on ajoute
                                     l'épaisseur d'une barre au
                                     dépassement
                < 0? int(niemeNote->OrdonneeMaximale(echelle) / echelle)
                      + EPAISSEUR_BARRE_NOTE - (HAUTEUR_INTERLIGNE << 2):
                     int(niemeNote->OrdonneeMaximale(echelle) / echelle)
                      - (HAUTEUR_INTERLIGNE << 2);
          if (depassementHaut <   // On met à jour s'il y a lieu les
                                     dépassements
                depassementHautNote)
                      depassementHaut = (int) depassementHautNote;
          if (depassementBas < depassementBasNote)
                depassementBas = (int) depassementBasNote;
     }
}

// Transformation du brouillon de barre en barre(s) graphique(s)
// Attention de n'utiliser barreDessus qu'à partir du deuxième niveau de
barre !
// Renvoie TRUE si tout c'est bien passé
BOOLEAN CBrouillonGraphiqueBarre::MiseAuPropre
```

```
      (CDecoupageMesure *mesure, CBrouillonGraphiqueBarre *barreDessus)
{     float echelle = mesure->      // Echelle de redimensionnement des
                                       graphiques musicaux
            Partition()->Echelle();
      CVecteur *graphiques          // Vecteur des graphiques de la mesure
            = mesure->Graphiques();
      long positionX,               // Coordonnée de l'objet graphique par
                                       rapport au point supérieur gauche de
                                       la partition
           positionY,
           ordonneePointDebut,      // Ordonnée des points de début et de
                                       fin
           ordonneePointFin;
      PNT pointDebut,               // Points de début et de fin de la barre
                                       (en coordonnées relative à l'objet
                                       graphique)
          pointFin;
      short decalageHampeHaute      // Décalage horizontal subi par la barre
                                       si les hampes sont hautes
            = short(LARGEUR_NOIRE * echelle) - 1;
      CGraphiqueBarreDeNote *barre; // Barre de note à ajouter aux
                                       graphiques de la mesure
      CVecteurIterateur             // Itérateur du vecteur des sous-barres
            barreSuivante(sousBarres);
      CBrouillonGraphiqueBarre *    // Sous-barre particulière
            sousBarre;
      if (!premiereNote)            // Si les notes extrémales ne sont pas
                                       encore initialisées,
      {     premiereNote =          // On retrouve la première note du n-
                                       olet
                  (CGraphiqueNote *) ChercheNiemeNote(graphiques,
                        indiceDebut, 1);
            derniereNote =          // On retrouve la dernière note du n-
                                       olet
                  (CGraphiqueNote *) ChercheNiemeNote(graphiques,
                        indiceDebut, nombreNotes);
      }
      ordonneePointDebut =          // On calcule l'ordonnée du point de
                                       début de la barre
            ((CGraphiqueNoteHampe *) premiereNote)->HampeHaute()?
            premiereNote->PositionY() + short(HAUTEUR_NOIRE * echelle) -
            ((CGraphiqueNoteHampe *)premiereNote)->HauteurHampe():
            premiereNote->PositionY() +
            ((CGraphiqueNoteHampe *) premiereNote)->HauteurHampe();
      ordonneePointFin =            // On calcule l'ordonnée du point de fin
                                       de la barre
            ((CGraphiqueNoteHampe *)derniereNote)->HampeHaute()?
            dernierNote->PositionY() + short(HAUTEUR_NOIRE * echelle) -
            ((CGraphiqueNoteHampe *)dernierNote)->HauteurHampe():
            dernierNote->PositionY() +
            ((CGraphiqueNoteHampe *) dernierNote)->HauteurHampe();
      if (niveau != 1)
            coefficientAngulaire =  // Le coefficient angulaire est le même
                                       que la barre au-dessus
                  barreDessus->CoefficientAngulaire();
      if (premiereNote ==           // Cas de la barre à une seule note
            dernierNote)
      {     pointDebut.h =          // On calcule les coordonnées du point
                                       de début de la barre
                  pointDebut.v = 0;
            pointFin.h =            // On calcule les coordonnées du point
                                       de fin de la barre
                  barreDessus->PremiereNote(graphiques) == premiereNote?
                  short(LARGEUR_BARRE_ORPHELINE * echelle):
```

```
            short(-LARGEUR_BARRE_ORPHELINE * echelle);
      pointFin.v =            // par rapport au point supérieur gauche
                              du graphique
            barreDessus->PremiereNote(graphiques) == premiereNote?
            short(coefficientAngulaire * 5):
            short(coefficientAngulaire * -5);
      positionX =             // On calcule les coordonnées du
                              graphique
            (((CGraphiqueNoteHampe *) premiereNote)->HampeHaute())?
            premiereNote→PositionX() + decalageHampeHaute:
            premiereNote->PositionX();
      positionY = ((CGraphiqueNoteHampe *)
            premiereNote)->HampeHaute()?
            ordonneePointDebut + long((HAUTEUR_INTERBARRE +
            EPAISSEUR_BARRE_NOTE) * echelle) * (niveau - 1):
            ordonneePointDebut - long((HAUTEUR_INTERBARRE +
            EPAISSEUR_BARRE_NOTE) * echelle) * (niveau - 1);
      if (!(barre = new       // On crée la barre de note graphique
            CGraphiqueBarreDeNote(positionX, positionY, &pointDebut,
                  &pointFin))) return FALSE;
      graphiques->push(barre);// On la note dans la mesure
      mesure->Partition()     // et dans la partition
            ->ObjetsGraphiques()->push(barre);
      return TRUE;
}
pointDebut.h = 0;             // On calcule les coordonnées du point
                              de début de la barre
pointDebut.v =                // par rapport au point supérieur gauche
                              du graphique
      coefficientAngulaire < 0?
      short(ordonneePointDebut - ordonneePointFin): 0;
pointFin.h = short(           // On calcule les coordonnées du point
                              de fin de la barre
      derniereNote->          // par rapport au point supérieur gauche
                              du graphique
      PositionX() - premiereNote->PositionX());
pointFin.v = coefficientAngulaire < 0?
      0: short(ordonneePointFin - ordonneePointDebut);
positionX =                   // On calcule les coordonnées du
                              graphique
      premiereNote->PositionX();
positionY = pointFin.v?
      (((CGraphiqueNoteHampe *) premiereNote)->HampeHaute()?
            ordonneePointDebut + long((HAUTEUR_INTERBARRE +
            EPAISSEUR_BARRE_NOTE) * echelle) * (niveau - 1):
            ordonneePointDebut - long((HAUTEUR_INTERBARRE +
            EPAISSEUR_BARRE_NOTE) * echelle) * (niveau - 1)):
      (((CGraphiqueNoteHampe *)premiereNote)->HampeHaute()?
            ordonneePointFin + long((HAUTEUR_INTERBARRE +
            EPAISSEUR_BARRE_NOTE) * echelle) * (niveau - 1):
            ordonneePointFin - long((HAUTEUR_INTERBARRE +
            EPAISSEUR_BARRE_NOTE) * echelle) * (niveau - 1));
if (((CGraphiqueNoteHampe *)  // on translate horizontalement la barre
                              si la hampe de la note est haute
      premiereNote)->HampeHaute())
            positionX += decalageHampeHaute;
if (!(barre = new             // On crée la barre de note graphique
      CGraphiqueBarreDeNote(positionX, positionY, &pointDebut,
            &pointFin))) return FALSE;
graphiques->push(barre);      // On la note dans la mesure
mesure->Partition()->         // et dans la partition
      ObjetsGraphiques()->push(barre);
while (sousBarre =            // On met au propre les sous-barres
      (CBrouillonGraphiqueBarre *) barreSuivante())
```

```
                    sousBarre->MiseAuPropre(mesure, this);
        return TRUE;
}
#endif
```

Annexe 7 : CMusique

Cette annexe contient le listing des fichiers CMusique.h, CMusique.cpp.

CMusique.h

```
// Nature: Définition des classes décrivant des notions purement musicales
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 29/10/96
// Modifications:
// Commentaires:

#ifndef CMusique_H
#define CMusique_H

#include "xvt.h"

typedef enum {cleSol, cleUt = 2,    // Les valeurs sont celles du code
                                       armature
      cleFa} nomCle;

class CMusiqueCle
{     private:                      // Implémentation en mémoire:
            nomCle nom;             //    Nom de la clé
            int ligne;              //    Ligne de positionnement de la clé
            short *                 //    Hauteur relative à la première
                                          ligne des altérations dans cette
                                          clé
                  hauteurAlterations;

      public:                       // Interface standard:
            CMusiqueCle             //    Constructeur
                  (nomCle, int);
            inline nomCle Nom()     //    Accès au nom de la clé
                  const {return nom;}
            inline void Nom         //    Changement du nom de la clé
                  (nomCle nomDesire)
                  {nom = nomDesire;}
            inline int Ligne()      //    Accès à la ligne de la clé
                  const {return ligne;}
            inline void Ligne       //    Changement de la ligne de la clé
                  (int ligneDesiree)
                  {ligne = ligneDesiree;}
            inline short            //    Accès à la hauteur d'une
                                          altération
                  HauteurAlteration(int alteration) const
                  {return *(hauteurAlterations + alteration + 7);}
};

class CMusiqueMesure
{     private:                      // Implémentation en mémoire:
            short numerateur;       //    Numérateur de la mesure
            short denominateur;     //    Dénominateur de la mesure
            BOOLEAN resumee;        //    Drapeau : la mesure est sous forme
                                          résumée (C, C barré, ...)

      public:                       // Interface standard:
            CMusiqueMesure          //    Constructeur
                  (short, short, BOOLEAN);
            inline short Numerateur //    Accès au numérateur de la mesure
                  () const{return numerateur;}
```

```
            inline short            //    Accès au dénominateur de la mesure
                  Denominateur() const {return denominateur;}
            inline BOOLEAN Resumee  //    Accès au drapeau de résumé de la
                                          mesure
                  () const {return resumee;}
            inline void Numerateur  //    Changement du numérateur de la
                                          mesure
                  (short numerateurDesire){numerateur = numerateurDesire;}
            inline void Denominateur//    Changement du dénominateur de la
                                          mesure
                  (short denominateurDesire)
                  {denominateur = denominateurDesire;}
            inline void Resumee     //    changement du drapeau de résumé de
                                          la mesure
                  (BOOLEAN resumeeDesire) {resumee = resumeeDesire;}
};

class CMusiqueHauteurNote
{     public:                       // Enumérations publiques:
            enum nom                //    Noms possibles d'une note
            {     Do = 0, Re, Mi, Fa, Sol, La, Si
            };

      private:                      // Implémentation en mémoire:
            short octave;           //    Octave de la note
            nom hauteur;            //    Hauteur de la note

      public:                       // Interface standard:
            CMusiqueHauteurNote     //    Constructeur
                  (short, nom);
            void Transpose(short);  //    Transposition
            void TransposeCleSol2   //    Transposition d'une note en son
                                          équivalent visuel clé de sol 2ème
                                          ligne
                  (CMusiqueCle *);
            int                     //    Calcul du nombre et de la position
                                          des lignes supplémentaires d'une
                                          note
                  CalculeLignesSupplementaires();
            int CalculeOrdonnee()   //    Calcul de l'ordonnée de la note
                                          par rapport à la première ligne de
                                          la portée
                  const;
            inline short Note()     //    Accès à la hauteur de la note
                  const {return hauteur;}
            inline short Octave()   //    Accès à la hauteur de la note
                  const {return octave;}
};

class CMusiqueNote
{     public:                       // Enumérations publiques:
            enum alterateur         //    Altérateur
            {     DoubleBemol = -2,
                  Bemol,
                  Becarre,
                  Diese,
                  DoubleDiese
            };

      private:                      // Implémentation en mémoire:
            CMusiqueHauteurNote     //    Hauteur de la note
                  hauteur;
            alterateur alteration;  //    Type d'altération
            int duree,              //    Durée de la note (nombre de ticks)
```

```
            rythmeVu;           //    Rythme de la note

      public:                   // Interface standard:
            CMusiqueNote        //    Constructeur
                  (const CMusiqueHauteurNote *, alterateur, int, int);
            inline              //    Accès à la hauteur de la note
                  CMusiqueHauteurNote *Hauteur() {return &hauteur;}
            inline alterateur   //    Accès à l'altération de la note
                  Alteration() const {return alteration;}
            inline int Duree()  //    Accès à la durée de la note
                  const {return duree;}
            inline int RythmeVu()   //    Accès au rythme de la note
                  const {return rythmeVu;}
            inline void Alteration  //    Changement l'altération de la note
                  (alterateur alterationNote)
                  {alteration = alterationNote;}
};

#endif
```

CMusique.cpp

```
// Nature: Implémentation des classes décrivant des notions purement
musicales
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 22/10/96
// Modifications:
// Commentaires:

#ifdef MUSIQUE

#include "MusiDef.h"
#include ConstantesMusique_i
#include CMusique_i

#define CLE_SOL1 0
#define CLE_SOL2 1
#define CLE_FA3 2
#define CLE_FA4 1
#define CLE_UT1 3
#define CLE_UT2 4
#define CLE_UT3 5
#define CLE_UT4 6
#define CLE_UT5 2

// Cas de la clé
// Constructeur
CMusiqueCle::CMusiqueCle(nomCle nomDesire, int ligneDesiree)
{     static short hauteurAlterationsPossibles[7][15] =
      {                             // Clé de sol 2ème ligne
            { -8,  10, -14,   4,  22,  -2,  16, 0,
              -5,  13, -11,   7,  25,   1,  19},
                                    // Clé de sol 1ère ligne ou clé de fa
                                       4ème ligne
            { 22,  -2,  16,  -8,  10, -14,   4, 0,
             -17,   1, -23,  -5,  13, -11,   7},
                                    // Cle de fa 3ème ligne ou clé d'ut 5ème
                                       ligne
            {  4,  22,  -2,  16,  -8,  10, -14, 0,
               7, -17,   1,  19,  -5,  13, -11},
            { 10, -14,   4,  22,  -2,  16,  -8, 0,
              13, -11,  13, -17,   7,  19,   1},
            { -2,  16,  -8,  10, -14,   4,  22, 0,
               1,  19,  -5,  13, -11,   7, -17},
```

```
            {-14,   4,  22,  -2,  16,  -8,  10, 0,
             -11,   7, -17,   1,  19,  -5,  13},
            { 16,  -8,  10, -14,   4,  22,  -2, 0,
              19,  -5,  13, -11,   7, -17,  -5}
      };

      nom = nomDesire;              // On initialise les données membres
      ligne = ligneDesiree;
      switch (nomDesire)            // On initialise le tableau des hauteurs
                                       suivant la clé désirée
      {     case cleSol:
                  hauteurAlterations// Clé de sol première ou 2ème ligne
                        = hauteurAlterationsPossibles
                              [ligneDesiree == 1? CLE_SOL1: CLE_SOL2];
                  break;
            case cleFa:
                  hauteurAlterations// Clé de sol première ou 2ème ligne
                        = hauteurAlterationsPossibles
                              [ligneDesiree == 3? CLE_FA3: CLE_FA4];
                  break;
            case cleUt:
                  switch (ligneDesiree)
                  {     case 1:     // Cas de la clé d'ut 1ère ligne
                        hauteurAlterations =
                              hauteurAlterationsPossibles[CLE_UT1];
                        break;
                        case 2:     // Cas de la clé d'ut 2ème ligne
                        hauteurAlterations =
                              hauteurAlterationsPossibles[CLE_UT2];
                        break;
                        case 3:     // Cas de la clé d'ut 3ème ligne
                        hauteurAlterations =
                              hauteurAlterationsPossibles[CLE_UT3];
                        break;
                        case 4:     // Cas de la clé d'ut 4ème ligne
                        hauteurAlterations =
                              hauteurAlterationsPossibles[CLE_UT4];
                        break;
                        default:    // Cas de la clé d'ut 5ème ligne
                        hauteurAlterations =
                              hauteurAlterationsPossibles[CLE_UT5];
                  }
      }
}

// Cas de la mesure
// Constructeur
CMusiqueMesure::CMusiqueMesure
      (short numerateurDesire, short denominateurDesire, BOOLEAN
resumeeDesire)
{     numerateur = numerateurDesire;
      denominateur = denominateurDesire;
      resumee = resumeeDesire;
}

// Cas de la hauteur de note
// Constructeur
CMusiqueHauteurNote::CMusiqueHauteurNote
      (short octaveDesire, nom hauteurDesiree)
{     octave = octaveDesire;
      hauteur = hauteurDesiree;
}

// Fonction de transposition
```

```
void CMusiqueHauteurNote::Transpose(short ecart)
{     short resultat =              // Résultat de la transposition
            octave * 7 + (short) hauteur + ecart;
      octave = resultat >= 0?       // On en déduit l'octave
             resultat / 7:
             resultat / 7 - 1;
      hauteur =                     // Et on en déduit la hauteur
            (nom) (resultat - 7 * octave);
}

// Fonction transposition de la note en son équivalent visuel
// en clé de sol 2ème ligne
void CMusiqueHauteurNote::TransposeCleSol2(CMusiqueCle *cle)
{     switch (cle->Nom())
      {     case cleSol:
                  if (cle->Ligne()  // Cas de la clé de sol 1ère ligne
                        == 1)
                        Transpose   // On transpose la note de 2 notes vers
                                       le bas
                              (-2);
                  break;
            case cleFa:             // Cas des clés de fa
                  Transpose(12);    // On transpose la note de 12 notes vers
                                       le haut
                  if (cle->Ligne()  // Si on est en clé de fa 4ème ligne,
                        == 4)
                        Transpose   // on la transpose encore de 2 notes
                                       vers le haut
                              (2);
                  break;
            case cleUt:             // Cas des clés d'ut
                  Transpose         // Plus la clé d'ut est haute, plus les
                                       notes sont hautes (visuellement, en
                                       clé de sol)
                        (2 * cle->Ligne());
      }
}

// Fonction de calcul du nombre de lignes supplémentaires de la note
int CMusiqueHauteurNote::CalculeLignesSupplementaires()
{     if (octave > 4 ||             // Cas de lignes supplémentaires au-
                                       dessus de la portée
            (octave == 4 && hauteur >= La))
            return                  // On calcule l'écart entre la note et
                                       la ligne de portée du fa4 (il est
                                       positif)
                  ((octave - 4) * 7 + hauteur - Fa) / 2;
      if (octave < 3 ||             // Cas de lignes supplémentaires en-
                                       dessous de la portée
            (octave == 3 && hauteur == Do))
            return                  // On calcule l'écart entre la note et
                                       la ligne de portée du mi3 (il est
                                       négatif)
                  ((octave - 3) * 7 + hauteur - Mi) / 2;
      return 0;
}

// Fonction de calcul de l'ordonnée du corps la note par rapport à
// la première ligne de la portée (la note est supposée en clé de
// sol 2ème ligne)
// Renvoie l'ordonnée du point supérieur gauche du corps de la note
int CMusiqueHauteurNote::CalculeOrdonnee() const
{     const int decalageDoGrave =   // Nombre d'interlignes avant d'arriver
                                       au Do(0)
```

```
            30;
      int nombreNotes = 7 * octave  // Nombre de notes entre le Do(0) et la
                                       note courante
            + hauteur;
      return (HAUTEUR_INTERLIGNE    // On en déduit l'ordonnée de la note
               * (decalageDoGrave - nombreNotes)) / 2;
}

// Cas de la note
// Constructeur
CMusiqueNote::CMusiqueNote
      (const CMusiqueHauteurNote *hauteurDesiree,
       alterateur alterationDesiree, int dureeDesiree, int rythmeDesire)
      : hauteur(*hauteurDesiree)
{     alteration = alterationDesiree;
      duree = dureeDesiree;
      rythmeVu = rythmeDesire;
}

#endif
```

Annexe 8 : CCachePixmap, CElementPixmap

Cette annexe contient le listing des fichiers CCacPMap.h, CCacPMap.cpp, CElePMap.h et CElePMapcpp.

CCapPMap.h

```
// Nature: Définition de la classe décrivant un cache contenant des pixmaps
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 18/10/96
// Modifications:
// Commentaires:

#ifndef CCachePixmap_H
#define CCachePixmap_H

#include "xvt.h"

#include "WaveDef.h"
#include CVecteur_i

class CCachePixmap
{     private:                        // Implémentation en mémoire:
            CVecteur cache;           //    Liste des pixmaps dans le cache
            int taille;               //    Taille du cache

      public:                         // Interface standard:
            CCachePixmap(int);        //    Constructeur
            ~CCachePixmap();          //    Destructeur
            XVT_PIXMAP Pixmap         //    Recherche de la pixmap indentifiée
                                      //    par son numéro de pixmap et son
                                      //    index de couleur
                  (int, COLOR, float, short, short, BOOLEAN);
            void VideCache();         //    Supprime les éléments dépassant du
                                      //    cache
};
#endif
```

CCacPMap.cpp

```
// Nature: Implémentation de la classe décrivant un cache de pixmaps
//       de musique dans une fenêtre XVT
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 18/10/96
// Modifications:
// Commentaires:
#ifdef MUSIQUE

#include "MusiDef.h"
#include CCachePixmap_i
#include CElementCachePixmap_i

#include "WaveDef.h"
#include CVecteurIterateur_i

// Constructeur
CCachePixmap::CCachePixmap(int tailleCache)
      : cache(tailleCache)
{     taille = tailleCache;         // On note la taille du cache
}

// Destructeur
```

```
CCachePixmap::~CCachePixmap()
{     cache.clearAndDestroy();      // On vide le cache
}

// Recherche de la pixmap dans le cache
// Renvoie NULL_PIXMAP en cas d'erreur
XVT_PIXMAP CCachePixmap::Pixmap
      (int numeroRessourceDessin, COLOR couleur, float echelle,
       short largeur, short hauteur,
       BOOLEAN enCouleur)
{     int anciennete;               // Ancienneté d'un élément dans le cache
      CVecteurIterateur             // Itérateur sur le cache
            prochainChamp(cache);
      CElementCachePixmap *         // Elément du cache
            elementCache;
      anciennete = 0;               // On part avec l'élément le plus récent
      while (elementCache =         // On parcourt le vecteur
            (CElementCachePixmap *) prochainChamp())
      {     if (elementCache->NumeroRessource() == numeroRessourceDessin &&
                elementCache->Couleur() == couleur &&
                elementCache->Echelle() == echelle)
                  break;            // On s'arrête quand on a trouvé
            anciennete++;
      }
      if (!elementCache)            // Si on ne l'a pas trouvé
      {     if (!(elementCache =    // Alors on crée l'élément correspondant
                  new CElementCachePixmap(couleur, echelle)))
            {     return NULL_PIXMAP;
            }
            if (!(elementCache->    // On l'initialise
                  Initialise(numeroRessourceDessin, largeur, hauteur)))
            {     delete elementCache;
                  return NULL_PIXMAP;
            }
//          if (cache.entries()     // Si le cache est saturé, on libère une
                                       derniere place
//                >= taille)
//                delete cache.pop();
            cache.insertAt          // On insère l'objet créé en première
                                       place du vecteur
                  (0, elementCache);
      }
      else
      {     cache.removeAt          // On retire l'élément de son ancienne
                                       place dans le cache
                  (anciennete);
            cache.insertAt          // On insère l'élément en tête du cache
                                       (le plus récent)
                  (0, elementCache);
      }
      return enCouleur?             // On renvoie la pixmap demandée
            elementCache->PixmapCouleur():
            elementCache->PixmapNoire();
}

// Supprime des éléments du cache pour que sa taille soit la taille
demandée lors de sa création
void CCachePixmap::VideCache()
{     while (cache.entries() >      // Tant que la taille du cache est trop
                                       grande
            taille)
      {     delete cache.pop();     // On supprime le dernier élément
      }
}
```

```
#endif
```

CElePMap.h

```
// Nature: Définition de la classe décrivant une pixmap élément d'un cache
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 18/10/96
// Modifications:
// Commentaires:

#ifndef CElementCachePixmap_H
#define CElementCachePixmap_H

#include "xvt.h"

#include "WaveDef.h"
#include CObjet_i

class CElementCachePixmap : public CObjet
{     private:                        // Implémentation en mémoire:
            int numeroRessource;      //    Numéro de ressource identifiant
                                            l'élément caché
            COLOR couleur;            //    Couleur utilisée dans la pixmap
                                            couleur
            float echelle;            //    Rapport entre les dimensions des
                                            pixmaps et des images
            XVT_PIXMAP pixmapNoire,   //    Pixmap en noir sur fond blanc
                       pixmapCouleur; //    Pixmap en couleur sur fond blanc

      public:                         // Interface standard:
            CElementCachePixmap       //    Constructeur
                  (COLOR, float);
            ~CElementCachePixmap();   //    Destructeur
            BOOLEAN Initialise        //    Initialisation
                  (int, short, short);
            inline XVT_PIXMAP         //    Accès à la pixmap noire
                  PixmapNoire() const {return pixmapNoire;}
            inline XVT_PIXMAP         //    Accès à la pixmap couleur
                  PixmapCouleur() const {return pixmapCouleur;}
            inline int                //    Accès au numéro de ressource
                  NumeroRessource() const {return numeroRessource;}
            inline float Echelle      //    Initialisation de l'échelle
                  () const {return echelle;}
            inline COLOR Couleur()    //    Accès à la couleur
                  const {return couleur;}
};
#endif
```

CElePMap.cpp

```
// Nature: Implémentation de la classe décrivant un pixmap élément d'un
//         cache de musique dans une fenêtre XVT
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 18/10/96
// Modifications:
// Commentaires:
#ifdef MUSIQUE

#include "xvt.h"

#include "Musique.h"

#include "MusiDef.h"
#include CElementCachePixmap_i
```

```
// Constructeur
CElementCachePixmap::CElementCachePixmap
      (COLOR couleurDesiree, float echelleDesiree)
      : CObjet()
{     echelle = echelleDesiree;     // On note les caractéristiques qui
identifient l'élément
      couleur = couleurDesiree;
      pixmapNoire = NULL_PIXMAP;    // On n'a pas encore de pixmaps
      pixmapCouleur = NULL_PIXMAP;
}

// Destructeur
CElementCachePixmap::~CElementCachePixmap()
{     if (pixmapNoire)              // Si on a des pixmaps, on les détruit
            xvt_pmap_destroy(pixmapNoire);
      if (pixmapCouleur) xvt_pmap_destroy(pixmapCouleur);
}

// Initialisation de l'élément
// Renvoie TRUE si tout s'est bien passé
BOOLEAN CElementCachePixmap::Initialise
      (int numeroRessourceDeclare, short largeur, short hauteur)
{     RCT rectangleSource,          // Rectangle de dimensions de l'image
          rectangleCible;           // Rectangle de dimensions de la pixmap
      XVT_IMAGE image;              // Image source
      numeroRessource =             // On note le numéro de ressource
            numeroRessourceDeclare;
      if (!(image =                 // On acquiert l'image par son numéro de
                                       ressource
            xvt_res_get_image(numeroRessourceDeclare))) return FALSE;
      xvt_rect_set                  // On initialise le rectangle source aux
                                       dimensions du dessin
            (&rectangleSource, 0, 0, largeur, hauteur);
      xvt_rect_set                  // On initialise le rectangle cible aux
                                       dimensions du dessin redimensionné
            (&rectangleCible, 0, 0, short(largeur * echelle),
            short(hauteur * echelle));
      if (!(pixmapNoire =           // On initialise le pixmap noir
            xvt_pmap_create(TASK_WIN, XVT_PIXMAP_DEFAULT,
             rectangleCible.right, rectangleCible.bottom, NULL)))
      {     xvt_image_destroy(image);
            return FALSE;
      }
      xvt_dwin_draw_image           // On dessine l'image dans la pixmap
                                       noire
            (pixmapNoire, image, &rectangleCible, &rectangleSource);
      xvt_image_set_clut            // On change la couleur noir par la
                                       couleur désirée pour la seconde
                                       pixmap
            (image, 0, couleur);
      if (!(pixmapCouleur =         // On initialise le pixmap couleur
            xvt_pmap_create(TASK_WIN, XVT_PIXMAP_DEFAULT,
             rectangleCible.right, rectangleCible.bottom, NULL)))
      {     xvt_image_destroy(image);
            return FALSE;
      }
      xvt_dwin_draw_image           // On dessine le dessin dans la pixmap
                                       couleur
            (pixmapCouleur, image, &rectangleCible, &rectangleSource);
      xvt_image_destroy(image);     // On détruit l'image
      return TRUE;
}
#endif
```

Annexe 9 : CRessources

Cette annexe contient le listing du fichier CRessources.java.

CRessources.java

```
// Nature: Définition de la classe décrivant un ensemble de ressources
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 07/01/97
// Modifications:
// Commentaires:

import java.util.Vector;
import java.awt.*;
import java.net.*;
import java.applet.*;
import CVecteurIterateur;

import java.awt.image.*;

public class CRessources
      implements Constantes
{
// Implémentation en mémoire:
      private Image cache[];        // Liste des images dans le cache
      private static String urlImages[] =
      {     "CleSol.gif",           // Clés 0
            "CleFa.gif", "CleUt.gif",
            "MesureC.gif",          // Mesures 3
            "MesureCBarre.gif",
            "Bemol.gif",            // Altérations 5
            "Diese.gif", "", "Becarre.gif",
            "Point.gif", "", "", "",// Point 9
            "MoustacheCrocheBas1.gif",// Moustaches 13
            "MoustacheCrocheBas2.gif", "MoustacheCrocheHaut1.gif",
            "MoustacheCrocheHaut2.gif",
            "Tempo.gif",            // Indication de tempo 17
            "Soupir.gif",           // Silences (sous-soupirs) 18
            "DemiSoupir.gif", "QuartSoupir.gif", "8iemeSoupir.gif",
            "16iemeSoupir.gif", "32iemeSoupir.gif", "64iemeSoupir.gif", "",
            "Pause.gif",            // Silences (sur-soupirs) 26
            "Pause.gif", "DoublePause.gif",
            "", "", "", "", "", "", // Silences pointés (sous-soupirs) 29
            "", "",
            "", "", "",             // Silences pointés (sur-soupirs) 37
            "Noire.gif",            // Notes 40
            "Blanche.gif", "Ronde.gif", "Carree.gif",
            "CercleRouge.gif",      // Avertissements 44
            "IconeMesure.gif", "IconeRythme.gif", "IconeAlteration.gif"
      };
      private MediaTracker pion;    // Surveillant de l'état de chargement
                                       des images
      private Applet appletMere;
      private boolean               // Etat de chargement des images du
                                       cache
            chargementTermine = false;

// Méthodes à usage privé:
// Attente de la terminaison du chargement des images
private boolean TermineChargement(int nombreEssais)
{     Object listeErreurs[];        // Liste des images où une erreur s'est
                                       produite lors du chargement
```

```
        int numeroErreur = 0;
        chargementTermine = false;    // Le chargement commence
        if (nombreEssais >= 10)       // Si le réseau est trop instable, on
                                         arrête
              return false;
        try
        {     pion.waitForAll();      // On attend la fin du chargement de
                                         toutes les images
        } catch (Exception e)
        {     System.out.println      // Exception si un thread vient
                                         interrompre le processus
                    ("exception CRessources1");
        }
        listeErreurs =                // On note les images erronnées
              pion.getErrorsAny();
        if (listeErreurs == null)     // S'il n'y a pas d'erreurs
        {     chargementTermine =     // Le chargement est terminé
                    true;
              return true;            // On a fini
        }
        numeroErreur = 0;
        try
        {     for (numeroErreur = 0;;numeroErreur++)
              {     pion.addImage(    // On resurveille le chargement de ces
                                         images
                          (Image) listeErreurs[numeroErreur], numeroErreur);
              }
        }
        catch (Exception e) {}
        return TermineChargement      // On attend à nouveau la fin du
                                         chargement
              (++nombreEssais);
}

// Méthodes:
// Constructeur
public CRessources(Applet appletAppelante)
{     cache = new Image[50];
      appletMere = appletAppelante;
      pion = new MediaTracker(appletAppelante);
}

// Demande de chargement d'une image
public void ChargeImage(int numeroRessource)
{     if (cache[numeroRessource - BASE_RESSOURCE] != null)
              return;                 // On a terminé si on charge déjà
                                         l'image
      try
      {     URL url = new URL       // On crée l'url de l'image
                    (appletMere.getCodeBase() + "images/" +
                    urlImages[numeroRessource - BASE_RESSOURCE]);
              Image image = appletMere.getImage(url);
              cache[numeroRessource - // On charge l'image et on la note dans
                                         le cache
                    BASE_RESSOURCE] = image;
              pion.addImage           // On demande au pion de surveiller
                                         cette image
                    (image, numeroRessource);
              pion.checkID            // On commence le chargement de cette
                                         image
                    (numeroRessource, true);
      } catch (Exception e) {}
}
```

```
// Recherche de l'image
public Image RetrouveImage(int numeroRessource)
{     if (chargementTermine == false)// Si le chargement n'est pas terminé
            if (!                    // on le termine
                  TermineChargement(0))
                  System.out.println("Erreur de chargement");
      try
      {     return cache             // On renvoie l'image demandée
                  [numeroRessource - BASE_RESSOURCE];
      }
      catch (Exception e)
      {     System.out.println("InfoMusic error.");
            System.out.println("Attempt to access an image outside the
                                    bounds of the cache array.");
            System.out.println("File : CRessources.java; Method
                                   RetrouveImage.");
            System.out.println("Resource number : " + numeroRessource);
            System.out.println("Please, be kind enough to report this
                  message and the error context to sturm@janus.u-
                  strasbg.fr.");
            System.out.println("Thanks.");
      }
      return null;
}

public Applet AppletMere()
{     return appletMere;
}
}
```

Annexe 10 : CalculeAngles

Cette annexe contient la méthode CalculeAngles de la classe CBrouillonGraphiqueLiaison.

CBrouillonGraphiqueLiaison.java

```
// Nature: Définition de la classe décrivant un brouillon d'une liaison
// Copyright: LOGI+, 6 rue de Stockholm, 67000 Strasbourg, France
// Création: Luc Lejoly, 13/01/97
// Modifications:
// Commentaires:

[...]

// Fonction privée de calcul des angles formés par les points de début et
// de fin de liaison dans le cercle
private void CalculeAngles
      (Rectangle carre, Point pointDebut, Point pointFin,
       boolean orientationDessus, Entier angleDebut, Entier angleArc)
{     double rayon = carre.width    // Rayon du cercle
            / 2.0;
      Point centre = new Point      // Centre du cercle
            ((int) (carre.x + rayon), (int) (carre.y + rayon));
      angleDebut.Valeur(            // On calcule les angles formés dans le
                                       cercle par le point de début et le
                                       point de fin
            orientationDessus?
                  (int) (Math.acos((pointDebut.x - centre.x) / rayon)
                        * 180 / Math.PI):
                  (int) (-Math.acos((pointDebut.x - centre.x) / rayon)
                        * 180 / Math.PI));
      int angleFin = orientationDessus?
            (int) (Math.acos((pointFin.x - centre.x) / rayon) * 180
                  / Math.PI):
            (int) (-Math.acos((pointFin.x - centre.x) / rayon) * 180
                  / Math.PI);
      angleArc.Valeur(angleFin -    // On calcule l'angle formé par les deux
                                       rayons
            angleDebut.Valeur());
}
[...]
}
```